Escuela
Politécnica
Superior

AirHockey VR

Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Autora:

Tamara Torrecillas Martínez

Tutor:

Juan Antonio Puchol García

Universitat d'Alacant
Universidad de Alicante

AirHockey VR

Videojuego de Realidad Virtual

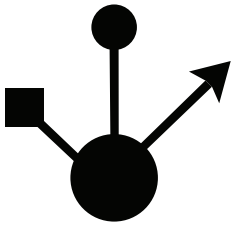
Autora

Tamara Torrecillas Martínez

Tutor

Juan Antonio Puchol García

Departamento de Ciencia de la computación e Inteligencia Artificial



Grado en Ingeniería Multimedia

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Julio 2021

Resumen

AirHockey VR es un videojuego multijugador de Realidad Virtual en el que se simula el tradicional juego de hockey sobre mesa. En este juego dos personas compiten sobre una mesa de aire, en la que emplearán un *pusher* para golpear el disco que deben meter en la portería contraria. Ganará, por tanto, el jugador que más goles haya anotado en el tiempo que dura la partida. Se trata de un juego multijugador, por lo que el usuario deberá enfrentarse a otros jugadores reales.

El juego cuenta además con un sistema propio de gestión de salas. En el *lobby* podrás seleccionar una de las salas ya existentes o crear tu propia sala. Una vez en partida deberás de estar muy atento a la misma, ya cualquier despiste puede suponer una ventaja para tu adversario.

Para llevar a cabo todas las fases del desarrollo del videojuego, además del diseño y la programación del mismo se ha llevado a cabo una mejora gráfica. Para ello se han diseñado y creado las distintas interfaces del juego y se han modelado y texturizado todos los elementos del entorno. Por último, para poder considerarlo un producto acabado, se han incorporado sonidos al juego, consiguiendo así una mejor experiencia de usuario.

Agradecimientos

Me gustaría agradecer a mi familia y amigos por el apoyo constante, ya no sólo durante el desarrollo de este proyecto sino durante toda mi etapa universitaria. En particular, agradecer a mi pareja por estar siempre dispuesto a ayudarme, por sus sabios consejos y por ser mi fuente de motivación.

Este trabajo cuenta con el apoyo del Ministerio de Ciencia, Innovación y Universidades (España), proyecto RTI2018-096219-B-100. Proyecto cofinanciado con fondos FEDER.

A mis abuelos Alfonso y Adela,
por ser mis ejemplos a seguir

Quien nunca ha cometido un error
nunca ha intentado nada nuevo

Albert Einstein.

Índice general

1. Introducción **1**

1.1. Motivación 1

1.2. Propuesta y Objetivos 2

1.3. Estructura 2

2. Estado del arte **4**

2.1. Introducción 4

2.2. Aplicaciones 5

2.2.1. Medicina 6

2.2.2. Entrenamiento 6

2.2.3. Educación 7

2.2.4. Cultura 8

2.2.5. Ocio y entretenimiento 9

2.2.6. Prevención de Riesgos Laborales 10

2.2.7. Ingeniería 10

2.2.8. Sector inmobiliario 11

2.3. Mercado actual 11

2.3.1. Modelos de gafas en el mercado 13

2.3.1.1. Gafas de RV para smartphones 13

2.3.1.2. Gafas de RV para PC 14

2.3.1.3. Gafas de RV para consola 18

2.3.1.4. Gafas autónomas 20

2.3.1.5. Comparativa 23

2.4. Videojuegos 25
2.4.1. Videojuegos en Oculus Quest 29

3. Metodología **33**
3.1. Mínimo producto viable 34

4. Documento de Diseño del Videojuego **35**
4.1. Características 35
4.2. Descripción 35
4.3. Ambientación y estilo 35
4.4. Jugabilidad 37
4.4.1. Objetivo 37
4.4.2. Mecánicas 37
4.5. Controles 38
4.6. Pantallas 39
4.7. Estados de Juego 44
4.8. Sistema sonoro 45
4.8.1. Música 45
4.8.2. Efectos de sonido 45

5. Herramientas **46**
5.1. Hardware 46
5.1.1. Oculus Quest 2 46
5.2. Software 47
5.2.1. Motor de juego 47
5.2.1.1. Unity 47
5.2.1.2. Unreal Engine 48
5.2.1.3. Comparativa 48
5.2.2. Oculus Integration 49
5.2.3. Sistema de multijugador 50
5.2.3.1. UNet 50
5.2.3.2. Photon PUN 2 50

5.2.3.3. Normcore . . . 51
5.2.3.4. Comparativa . . . 52
5.2.4. MongoDB Atlas . . . 53
5.2.5. Blender . . . 54
5.2.6. GIMP . . . 54
5.2.7. Inkscape . . . 54
5.2.8. Audacity . . . 54

6. Desarrollo **55**
6.1. Configuración del proyecto . . . 55
6.2. Pruebas de los sistemas de multijugador . . . 59
6.2.1. Photon PUN 2 . . . 59
6.2.2. Normcore . . . 60
6.3. Implementación . . . 61
6.3.1. Escena Lobby . . . 62
6.3.2. Escena Air Hockey . . . 67
6.4. Mejora gráfica e incorporación de audio . . . 75
6.4.1. Iluminación . . . 75
6.4.2. Modelado . . . 75
6.4.3. Interfaces . . . 83
6.4.4. Música y efectos de sonido . . . 89

7. Conclusiones **91**
7.1. Trabajo futuro . . . 92

Referencias **93**

Lista de Acrónimos y Abreviaturas **99**

A. Anexo I - Créditos **100**
A.1. Imágenes . . . 100
A.2. Modelados . . . 102
A.3. Sonidos . . . 102

B. Anexo II - Descarga APK del videojuego **104**

B.1. Instalación vía SideQuest VR 104

B.2. Instalación vía Android SDK Platform Tools 105

B.3. Ejecución de la aplicación 105

Índice de figuras

2.1. Las 3 I's de la RV 5
2.2. Neuro-rehabilitación con RV 6
2.3. Uso de la RV para el entrenamiento militar 7
2.4. Niño haciendo uso de la RV para aprender 8
2.5. Uso de la RV en las visitas a museos 9
2.6. Dos personas jugando a un videojuego en RV 9
2.7. Ejemplo de prevención de riesgos laborales con RV 10
2.8. Ejemplo del uso de la RV en el campo de la ingeniería 11
2.9. Visita de una vivienda mediante RV 11
2.10. Mercado RV entre 2016-2020 clasificado por plataforma 12
2.11. Gafas de RV para smartphones 14
2.12. Oculus Rift S 15
2.13. Gafas HTC Vive 16
2.14. Valve Index 17
2.15. Gafas compatibles con Windows Mixed Reality 18
2.16. Nintendo Switch VR Labo - Pack completo 19
2.17. PlayStation VR 19
2.18. Oculus Quest 2 20
2.19. Ejemplo de impresiones 3D para mejorar la correa de Oculus Quest 2 22
2.20. Ejemplo de impresiones 3D para imitar armas 22
2.21. Ejemplo de impresiones 3D para deportes 23
2.22. Ejemplo de impresiones 3D para soporte de las Oculus Quest 2 23
2.23. Half-Life: Alyx 25
2.24. Beat Saber 26

2.25. Astro Bot Rescue Mission 26
2.26. Tetris Effect 27
2.27. SuperHOT 27
2.28. Arizona Sunshine 28
2.29. Trover Saves the Universe 28
2.30. Eleven Table Tennis 29
2.31. Sports Scramble 30
2.32. The Climb 30
2.33. Hyper Dash 31
2.34. Cubism 31
2.35. Puzzling Places 32

4.1. Ejemplo de habitación Gamer 36
4.2. Casco de hockey profesional 37
4.3. Mandos de Oculus Quest 38
4.4. Mandos de Oculus Quest 2 con los botones identificados 39
4.5. Pantalla para introducir nombre 40
4.6. Pantalla con la lista de salas 40
4.7. Pantalla para crear sala 41
4.8. Pantalla de espera después de crear la sala 42
4.9. Cuenta atrás para comenzar la partida 42
4.10. Durante la partida 43
4.11. Fin de la partida 43
4.12. Diagrama de flujo 44

5.1. Ejemplo de la escena de prueba con dos usuarios 53

6.1. Módulos necesarios para compilar para el Oculus Quest 55
6.2. Pestaña para configurar el sistema operativo para el que vamos a compilar . . 56
6.3. Asset Store de Unity una vez descargado el paquete Oculus Integration . . . 57
6.4. Ventana de XR Plug-in Management con la opción de Oculus seleccionada . . 57
6.5. configuración de OVRPlayerController para poder coger y mover objetos . . . 58

6.6. Prototipo de la pantalla de PlayerNameUI dentro del juego 64
6.7. Prototipo de la pantalla de RoomsAvailableUI dentro del juego 65
6.8. Prototipo de la pantalla de CreateRoomUI dentro del juego 65
6.9. Prototipo de la pantalla de WaitingUI dentro del juego 66
6.10. Prototipo de la pantalla de ConnectionFailsUI dentro del juego 67
6.11. Prototipo de las interfaces ScoreObjects y TimeObjects dentro del juego . . . 69
6.12. Prototipo de la interfaz EndScreens dentro del juego 69
6.13. Nombre del jugador encima de su cabeza dentro del juego 73
6.14. Prototipo de la interfaz EndScreens cuando el otro jugador abandona la partida dentro del juego 74
6.15. Render de la escena de Lobby. Cámara enfocando a los sofás. 76
6.16. Render de la escena de Lobby. Cámara enfocando a la puerta de room. 76
6.17. Render de la escena de Lobby. Cámara enfocando a la puerta de exit. 77
6.18. Render de la escena de Lobby. Cámara enfocando al aparador. 77
6.19. Cartel del juego 78
6.20. Render de la escena de Air Hockey. Cámara enfocando a las máquinas arcades. 79
6.21. Render de la escena de Air Hockey. Cámara enfocando al sofá. 79
6.22. Render de la escena de Air Hockey. Cámara enfocando a los pósteres. 80
6.23. Render de la escena de Air Hockey. Cámara enfocando a las lejas. 80
6.24. Render de la escena de Air Hockey. Cámara enfocando al aparador. 81
6.25. Render de la escena de Air Hockey. Cámara desde la perspectiva del jugador 1. 81
6.26. Render de la escena de Air Hockey. Cámara desde la perspectiva del jugador 2. 82
6.27. Casco de Hockey modelado por Andrey Novichkov 82
6.28. Disco fracturado 83
6.29. Color beige y celeste empleados en la interfaz 83
6.30. Tipos de botones diseñados y su funcionalidad 84
6.31. Interfaz de PlayerNameUI 84
6.32. Interfaz de RoomsAvailableUI 85
6.33. Interfaz de CreateRoomUI 85
6.34. Interfaz de WaitingUI 86

6.35. Interfaz de ConnectionFailsUI 86
6.36. Interfaz de TimeObjects con la cuenta atrás activada 87
6.37. Interfaz de ScoreObjects y TimeObjects con la partida empezada 88
6.38. Interfaz de EndScreens con la partida finalizada 88
6.39. Interfaz de EndScreens cuando un jugador abandona la sala 89

B.1. Menú superior de la aplicación SideQuest VR con la opción de Instalar APK marcada 104
B.2. APK de AirHockey VR dentro de la carpeta de Platform Tools 105
B.3. Menú de Aplicaciones de Oculus Quest con la opción de Orígenes desconocidos señalada 106

Índice de tablas

2.1. Comparativa de las características técnicas de los dispositivos de VR 24

Índice de Códigos

B.1. Comando para instalar APK 105

1. Introducción

En los últimos años hemos podido observar que los usos de la Realidad Virtual no se limitan únicamente al entretenimiento. Esta tecnología se está expandiendo a otros campos como pueden ser la educación, la medicina o el entrenamiento militar [1].

Si bien es cierto que esta tecnología se encuentra en su mejor momento, debido a los precios asequibles de los nuevos cascos de Realidad Virtual como Oculus Quest 2, analistas como George Jijiashvili consideran que la falta de juegos y aplicaciones atractivas hacen imposible la aceptación masiva de esta tecnología [2].

Por otra parte, un estudio dirigido por la Universidad de Portsmouth examinó el uso de la Realidad Virtual en individuos de todas las edades durante el confinamiento [3]. Los resultados de dicho estudio indican que el uso de esta tecnología tuvo un impacto positivo en su salud mental, ya que les permitía jugar, hacer ejercicio, meditar, socializar y ver películas. Consideran que estos resultados podrían suponer un impulso hacia la implementación de estrategias basadas en Realidad Virtual para apoyar la salud física y mental de la población, incluso después de superar el Covid-19.

Como desarrolladores de esta tecnología, todavía hay una gran cantidad de desafíos por superar. Además, la creciente popularidad y demanda de estos sistemas hace que la innovación continúe y cada vez sean más los usos y tendencias de la Realidad Virtual [4].

1.1. Motivación

La idea de este Trabajo de Fin de Grado (TFG) surge como una propuesta propia al Departamento de Ciencia de la computación e Inteligencia Artificial de la Universidad de Alicante.

El motivo de proponer este proyecto se debe a que me permite aprender tecnologías que

no hemos visto durante el transcurso de la carrera. Además, me ofrece la oportunidad de mejorar mis habilidades como desarrolladora de videojuegos, permitiéndome hacerme cargo de todas las fases que conllevan el desarrollo de este tipo de proyectos.

Por último, la selección de la tecnología de Realidad Virtual para el proyecto se debe a la creciente popularidad de la misma. Este campo proporciona muchas posibilidades de empleo, por lo que resulta importante ampliar los conocimientos sobre la misma.

1.2. Propuesta y Objetivos

El principal propósito de este proyecto es desarrollar un videojuego multijugador para Realidad Virtual. Para ello se deberá estudiar los diversos motores que admiten esta tecnología y seleccionar el que mejor se adecúe a nuestro proyecto.

Una vez seleccionado el motor, se deberá investigar cuál es el mejor sistema de multijugador para la temática elegida. Esta investigación consistirá tanto en el estudio de los diversos sistemas existentes en el mercado como en la realización de pruebas para comprobar su funcionamiento.

El siguiente paso será el diseño y desarrollo de las mecánicas de juego. Estas deberán ser sencillas para así poder alcanzar un mayor público objetivo y para disminuir el tiempo necesario para aprender a manejar el entorno.

Una vez implementado el videojuego, se deberá realizar una mejora gráfica del mismo. Esta mejora consistirá tanto en el modelado y texturizado del entorno como en el diseño de las distintas interfaces. Además, se deberán incluir los sonidos correspondientes al videojuego.

1.3. Estructura

Esta memoria está estructurada en siete capítulos. El primer capítulo contiene una introducción a la situación actual de la Realidad Virtual, los objetivos de este proyecto y finalmente la estructura de la memoria.

El segundo capítulo describe el estado del arte. En este se realiza una introducción de qué es la Realidad Virtual, sus aplicaciones, el mercado actual y los videojuegos más destacados para la misma.

El tercer capítulo explica la metodología empleada en el desarrollo de este proyecto.

En el cuarto capítulo encontramos el documento de diseño del Videojuego, en el que se explican las características del mismo, la jugabilidad, los controles, las distintas pantallas y estados del juego y el sistema sonoro.

El quinto capítulo explica con detalle las herramientas utilizadas para el desarrollo del TFG, tanto hardware como software.

El sexto capítulo describe todo el proceso de desarrollo, desde las pruebas iniciales hasta la mejora gráfica final.

El séptimo y último capítulo presenta las conclusiones del TFG y se indican las características adicionales que se podrían implementar en un futuro.

2. Estado del arte

2.1. Introducción

Si buscamos la definición de Realidad Virtual (RV) en Internet, podremos observar que cada página nos ofrece una definición distinta de la misma. A pesar de no existir una definición estandarizada de qué es la RV, todas ellas comparten unas características comunes. Gracias a ello podemos establecer las características que debería de tener un sistema de RV [5]:

- Entorno generado por ordenador (mundo virtual).
- Simulación en tiempo real.
- Interacción a través de diferentes canales sensoriales.
- Retroalimentación sensorial.
- Inmersión física o psicológica.

Este mundo virtual podrá simular un mundo real o imaginario. Normalmente estos mundos virtuales son entornos tridimensionales, con el objetivo de manipular los sentidos del usuario, principalmente la visión, para que perciba ese entorno virtual como real.

Por lo que respecta a que sea en tiempo real, nos referimos a que el mundo es dinámico y reacciona con el usuario. Es decir, un buen sistema de RV requiere una interacción y una retroalimentación inmediatas. Esto supone que los requerimientos computacionales de los sistemas de RV sean muy altos.

Por otro lado, tenemos lo que se conoce como las 3 I's de la RV. Los sistemas de RV son una integración de estas tres características que se muestran en la figura 2.1.

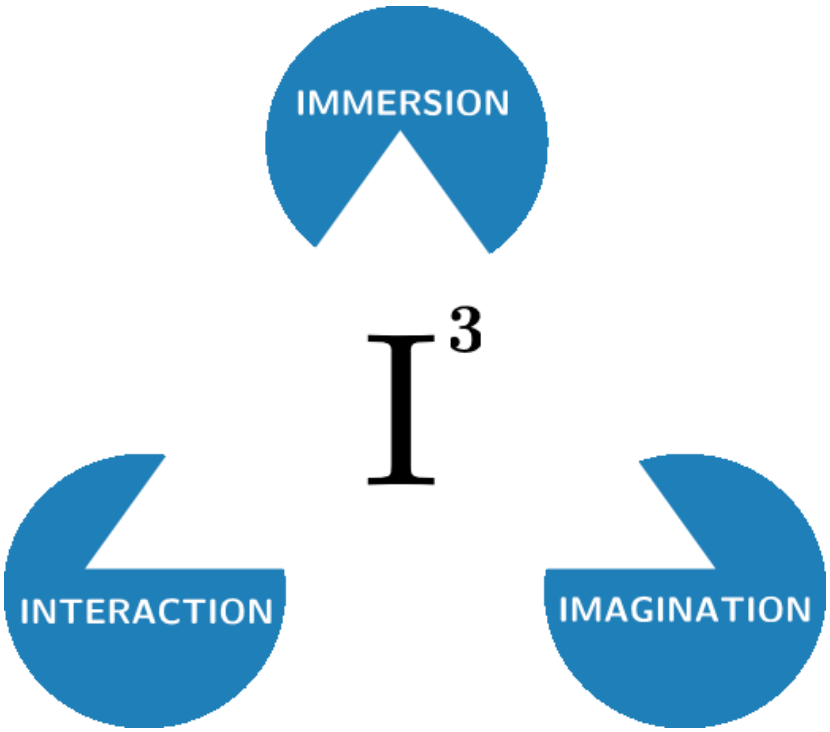


Figura 2.1: Las 3 I's de la RV

Estas características las podemos definir como:

- **Inmersión:** medios y habilidad del equipo diseñador para involucrar al usuario en la escena. Puede ser inmersión física o mental.

- **Interacción:** para que un sistema de RV parezca auténtico, éste debe responder a las acciones del usuario.

- **Imaginación:** capacidad mental para percibir cosas que no existen.

2.2. Aplicaciones

El uso de la RV se está extendiendo y cada vez son más los campos que emplean esta tecnología. En este apartado explicaremos algunos de estos usos [1].

2.2.1. Medicina

El uso de la RV en este campo supone un gran potencial. Esta se puede emplear para la formación de estudiantes y médicos, el tratamientos de fobias y traumas psicológicos, la planificación preoperatoria, la asistencia durante una cirugía y para rehabilitación.

Figura 2.2: Neuro-rehabilitación con RV

2.2.2. Entrenamiento

La RV ha sido de ayuda para el entrenamiento de profesionales militares, ya que les permite mejorar sus habilidades y capacidades sin necesidad de estar en el campo de batalla. Estos entrenamientos son menos costosos y permiten simular diversas situaciones y terrenos a los que los militares deben de enfrentarse.

Figura 2.3: Uso de la RV para el entrenamiento militar

La RV se puede emplear para simulaciones de vuelo para el ejército del aire y simulaciones de conducción para vehículos de tierra como tanques o camiones. Estas simulaciones permiten ahorrar costes, garantizan una mayor seguridad y evitan la contaminación que se produciría en entrenamientos reales.

2.2.3. Educación

Una buena forma de interiorizar conceptos es mediante juegos y actividades. Gracias a la RV disponemos de una herramienta de aprendizaje divertida, donde los niños podrán aprender jugando. No obstante, las posibilidades de esta en el ámbito de la educación van más allá y proporcionan ventajas a estudiantes de todas las edades, no solo niños. Esta se puede emplear en ámbitos universitarios para que estudiantes de ingeniería puedan diseñar modelos de arquitecturas o para la formación de estudiantes de medicina, por ejemplo.

Figura 2.4: Niño haciendo uso de la RV para aprender

2.2.4. Cultura

Cada vez son más las aplicaciones que nos permiten realizar recorridos virtuales por cualquier parte del mundo. Los objetivos de estas aplicaciones no es sustituir los viajes físicos, sino incentivar a las personas a viajar a esos lugares.

Además, está creciendo su uso en museos, tanto para hacer visitas virtuales desde casa como para complementar las visitas que se realizan en este. Esto último consigue aportar dinamismo a las visitas, mejoran la experiencia de los visitantes y atraer a otros nuevos. Algunas de las técnicas que se emplean son crear un *story telling* de forma que la visita al museo se convierta en un juego, mostrar un espacio histórico que ya no existe en la actualidad o interactuar con objetos valiosos sin dañar los objetos reales.

Figura 2.5: Uso de la RV en las visitas a museos

2.2.5. Ocio y entretenimiento

La industria de los videojuegos es la que más aplicaciones de RV desarrolla. Cada vez son más las empresas que se suman a esta tendencia y pretenden diferenciarse de los videojuegos tradicionales. Gracias a los dispositivos de RV como Oculus VR o PlayStation VR, los videojuegos han podido dar un salto cualitativo.

Figura 2.6: Dos personas jugando a un videojuego en RV

No obstante, estos dispositivos nos permiten disfrutar de otras experiencias, como puede ser ver un vídeo como si nos encontráramos en el cine, recrear la experiencia de escalar una montaña, asistir a conciertos desde casa o acceder a salas sociales donde poder conocer gente.

2.2.6. Prevención de Riesgos Laborales

El uso de la RV en este campo nos permite evaluar de forma segura las reacciones de los usuarios ante determinados riesgos y corregir estas reacciones. Entre sus ventajas se encuentra la focalización de la atención por parte del usuario y la cantidad de información que se puede capturar de estas simulaciones y que nos permiten adoptar medidas en caso de detectar errores [6].

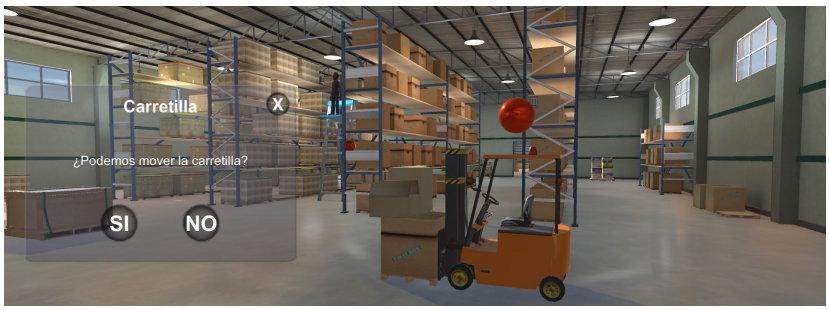


Figura 2.7: Ejemplo de prevención de riesgos laborales con RV

2.2.7. Ingeniería

Gracias al uso de la RV en este sector, se pueden recrear proyectos, diseños, maquetas, coches o edificaciones antes de que sean creados. Esto nos da la posibilidad de comprobar si ha habido algún error en su diseño y realizar mejoras y cambios para asegurarnos de que no habrá ningún problema a la hora de crearlos.

Figura 2.8: Ejemplo del uso de la RV en el campo de la ingeniería

2.2.8. Sector inmobiliario

La RV cada vez es más común en este sector para realizar visitas virtuales a casas que están sin construir o mostrar como una casa quedaría amueblada. Estas visitas virtuales hacen que el usuario se sienta más atraído por la vivienda que al mostrar unos planos de la misma.

Figura 2.9: Visita de una vivienda mediante RV

2.3. Mercado actual

Como podemos observar en el gráfico de la figura 2.10, el crecimiento del mercado de RV ha ido creciendo poco a poco hasta 2020, donde podemos ver un gran crecimiento sobretodo en las

plataformas de consola y PC. En el caso de los móviles, en 2020 ha crecido con respecto a los años anteriores, pero este crecimiento no es tan destacado como en las otras dos plataformas.

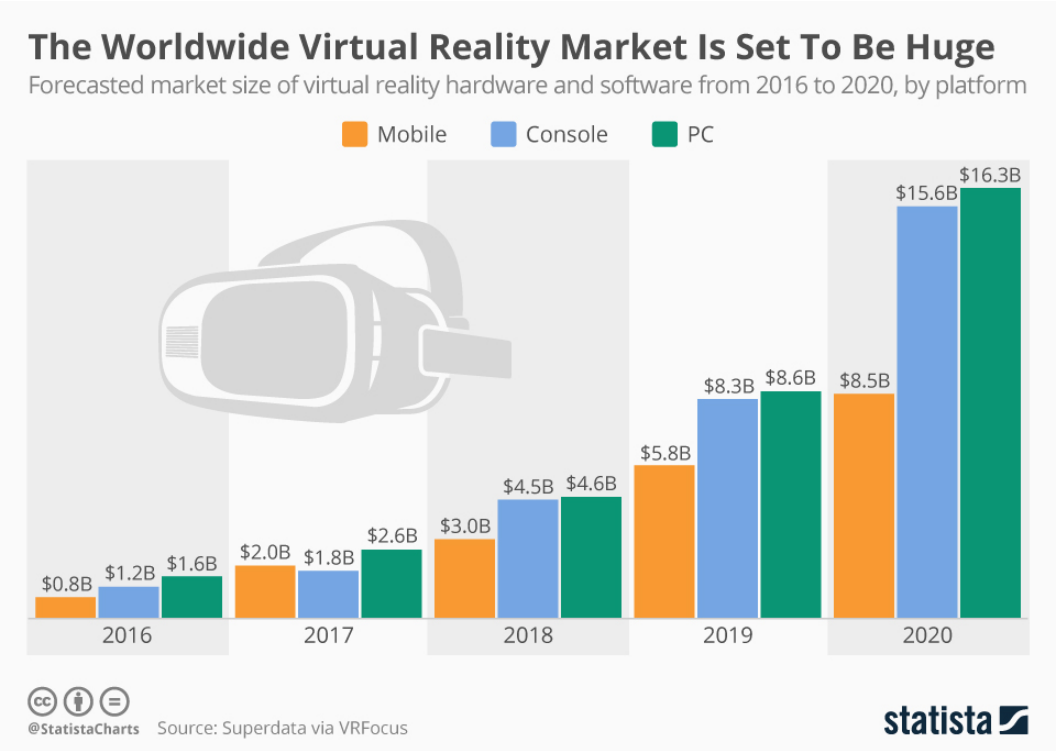


Figura 2.10: Mercado RV entre 2016-2020 clasificado por plataforma

GlobalData[1] espera que el mercado de la RV alcance un valor de 28.000 millones de dólares en 2030, habiendo crecido a una tasa de crecimiento anual compuesta (CAGR)[2] del 13% durante ese período.

El artículo de Computer World [7] nos indica que según el estudio *Virtual Reality - Thematic Research* de GlobalData, las principales tendencias en materia de tecnología en el campo de la RV son:

- **Silicio personalizado:** el desarrollo de la RV ha sido posible gracias a los avances en la tecnología de semiconductores. No obstante, se está estudiando el uso del silicio construido expresamente para ello como alternativa a lo que se venía empleando.

[1] GlobalData es una empresa de consultoría y análisis de datos, con sede en Londres.
[2] La tasa de crecimiento anual compuesto (o CAGR por sus siglas en inglés *Compound Annual Growth Rate*) nos permite conocer la tasa de retorno que alcanza una determinada inversión durante un periodo concreto.

- **Audio 3D:** para lograr que el usuario disfrute de una experiencia realmente inmersiva es necesario sincronizar correctamente las imágenes con el audio 3D.

- **Inteligencia artificial:** el uso de esta en las aplicaciones de RV permite mejorar la inteligencia de los NPCs y conseguir una mayor inmersión.

- **Plataformas conversacionales:** la falta de alternativas creíbles supone una oportunidad de negocio.

- **Cloud:** permitirá una mayor escalabilidad a los proveedores de RV.

- **5G:** va a suponer una baja latencia, una alta densidad y una mayor fiabilidad. Permitirá reducir el riesgo de atenuación de la transmisión.

2.3.1. Modelos de gafas en el mercado

En este apartado clasificaremos los distintos modelos de gafas de RV en función del hardware complementario que necesitan estas: un smartphone, un PC, una consola o si son autónomas [8].

2.3.1.1. Gafas de RV para smartphones

Estas gafas integran nuestro smartphone para usarlo como pantalla, carecen de sistemas de detección (con algunas excepciones) y las características técnicas dependen de nuestro móvil. Son menos potentes e inmersivos que el resto de dispositivos, pero su precio es mucho más bajo.

Por lo que respecta a los videojuegos y las aplicaciones para estas gafas, únicamente tendremos que acceder a Google Play o App Store para descargarlos en nuestro smartphone.

Entre los modelos que podemos encontrar en el mercado se encuentran las **Carl Zeiss VR One Plus** (29,99 euros) [9], diseñadas en colaboración con Oculus, y las **Google CardBoard** (podemos encontrar varios modelos de las mismas, el más barato a 7,40 euros) [10]. Estas últimas, a diferencia de las otras, son de cartón y hay que montarlas para poder utilizarlas.

(a) Carl Zeiss VR One Plus **(b)** Google CardBoard

Figura 2.11: Gafas de RV para smartphones

2.3.1.2. Gafas de RV para PC

El ordenador es, por el momento, el dispositivo por excelencia a la hora de disfrutar de la RV gracias a la potencia que este nos proporciona. No obstante, es importante saber que para poder emplear RV en un ordenador es necesario que este la soporte, por lo que deberemos de cumplir con determinados requisitos según el modelo de gafas que elijamos.

Podemos entender estas gafas como si fueran el monitor de nuestro ordenador, donde vamos a visualizar el juego, mientras que el ordenador será donde este se ejecute. Es por ello que el hardware del ordenador es tan importante.

Entre los distintos modelos que podemos encontrar cabe destacar los siguientes:

- **Oculus Rift S:** cuenta con una óptica mejorada y mayor nitidez en comparación con sus predecesores. No tiene sensores externos, ya que los integra en el propio dispositivo y viene con dos controladores Touch. Además, dispone de una herramienta que nos permite comprobar si nuestro ordenador es compatible con estas.

  Oculus cuenta con una tienda propia, que dispone de un amplio catálogo de juegos y aplicaciones.

Figura 2.12: Oculus Rift S

El pack Oculus Rift S (569,99 euros) incluye visor, dos controladores, pilas, cables y salida de vídeo [11].

- **HTC Vive:** encontramos actualmente en el mercado tres modelos distintos: HTC Vive Pro, HTC Vive Cosmos y HTC Vive Cosmos Elite. Destacan por la experiencia inmersiva que ofrecen y su amplia gama de juegos, puesto que además del catálogo de StremVR de Valve podremos jugar al catálogo de Oculus.

  Para un correcto funcionamiento de HTC Vive, se recomienda un ordenador potente. Para ello Valve ha preparado un programa que testea tu PC y te informa de si las gafas funcionarán sin problemas o de si deberás de cambiar alguna pieza de tu PC para que las HTC Vive funcionen correctamente.

(a) HTC Vive Pro

(b) HTC Vive Cosmos

(c) HTC Vive Cosmos Elite

Figura 2.13: Gafas HTC Vive

Las HTC Vive Cosmos (879 euros) [12] cuenta con seis cámaras que ofrecen la tecnología de seguimiento, por lo que no necesitan estaciones base. Por otro lado, HTC Vive Pro (1370 euros) [13] y HTC Vive Cosmos Elite (999 euros) [14] si que incluyen estaciones base. Cabe destacar además que las HTC Vive Pro disponen de *traking* de ojos.

- **Valve Index:** es la ambiciosa propuesta de Valve para los usuarios más exigentes. Dispone de varios kits, en concreto el kit completo puede resultar interesante si empezamos de cero. Este cuesta 1079 euros e incluye un visor, dos mandos, dos estaciones base y el juego *Half-Life: Alyx* [15].

Figura 2.14: Valve Index

De nuevo, para saber si tu ordenador es compatible, Valve pone a tu disposición una herramienta que te permitirá comprobarlo.

- **Windows Mixed Reality:** es la propuesta de Microsoft para disfrutar de la realidad mixta en Windows 10. Este nos proporciona una mezcla entre la RV y la aumentada. Para poder disfrutar de esta experiencia será necesario un ordenador lo suficientemente potente, unas gafas certificadas por Microsoft y un controlador para poder moverte por el entorno. Windows Mixed Reality permite ejecutar en dos versiones: una más sencilla y otra de más calidad. Este también nos proporciona una herramienta para saber si nuestro ordenador es compatible, denominada Windows Mixed Reality PC Check.

  En cuanto a las aplicaciones y juegos, Microsoft cuenta con una sección de juegos adaptados en su tienda, con una app que permite acceder a los juegos de Steam y con una serie de aplicaciones sociales y de entretenimiento. Por último, algunas de las gafas compatibles con la realidad mixta de Windows son las HoloLens 2 (3849 euros) [16] o las HP Reverb G2 (699 euros) [17].

(a) Hololens 2

(b) HP Reverb G2

Figura 2.15: Gafas compatibles con Windows Mixed Reality

2.3.1.3. Gafas de RV para consola

Actualmente, solo dos consolas permiten acceder a ella de forma nativa, aunque la experiencia que estas proporcionan es muy diferente: la Nintendo Switch y la PlayStation 4. Existe la posibilidad también de acceder a los juegos de Xbox mediante Oculus Rift, pero bajo ciertas condiciones.

Los modelos de gafas que podemos encontrar en este campo son principalmente:

- **Nintendo Switch VR Labo:** la propuesta de estos se encuentra más atrasada en comparación con el resto de propuestas en materia de videojuegos para consolas y PC. Estas gafas son de cartón, similares a las Google CardBoard, con la diferencia de que, en lugar de integrar el smartphone, en estas se integra la Switch.

  Claramente, para poder emplear estas será necesario disponer de la consola Nintendo Switch. El kit Nintendo Switch VR Labo cuesta 39,95 euros e integra el propio visor, un desintegrador Joy-con y un set de actividades que cuenta con la opción de ampliarlo con dos sets extra. El kit que incluye estas dos extensiones cuesta 79,90 euros. Su bajo precio en comparación con otras propuestas para videojuegos se debe a la sencillez de las gafas.

Figura 2.16: Nintendo Switch VR Labo - Pack completo

Cabe destacar que actualmente el pack básico y el completo se encuentran agotados. No obstante, podemos encontrar los dos packs de extensiones por 19,95 euros cada uno [18].

- **PlayStation VR:** permiten experimentar la RV de forma asequible si lo comparamos con otras propuestas para jugar. En este caso no tendremos que preocuparnos de los requerimientos del visor, ya que únicamente necesitaremos tener la PS4 con su respectivo mando Dualshock para jugar.

Figura 2.17: PlayStation VR

Podemos encontrar el dispositivo a la venta con un kit que incluye la cámara, el visor y algún juego. El precio de este kit suele ser 299,99 euros [19]. Para determinados juegos puede interesar adquirir los accesorios PlayStation Move o el mando pistola de PS VR.

El punto débil de estas gafas se debe a que, al integrar tantos cables, en la práctica inducen a emplearlas sentados en detrimento de la experiencia de uso.

2.3.1.4. Gafas autónomas

La principal propuesta en el mercado de gafas autónomas viene de la mano de la empresa Oculus. Las gafas Oculus Quest 2 no necesitan que las conectes a un smartphone u ordenador, sino que con las propias gafas y los controladores será suficiente. Estas cuentan con un sistema operativo basado en Android.

Figura 2.18: Oculus Quest 2

A pesar de toda la tecnología que se encuentra dentro del visor, no son un dispositivo pesado. Además, al no emplear cables permitirán una mayor movilidad al usuario. Son una buena opción para aquellos que quieran llevarlas de viaje.

Tanto Oculus Quest como Oculus Quest 2 disponen de *traking* de manos. Estas últimas son menos pesadas y tienen algunas mejoras tecnológicas con respecto a su predecesor. No obstante, son bastantes las quejas con respecto a la nueva correa del visor de Oculus Quest 2 por lo que han lanzado por separado una correa más parecida a la de su predecesor y que cuesta unos 49 euros [20].

En cuanto al catálogo de juegos y aplicaciones, podemos encontrar una gran variedad en la propia tienda de Oculus. Además, podemos encontrar muchos juegos indies desarrollados para

Oculus en la tienda SideQuest. En esta página podemos encontrar tanto juegos gratuitos, de pago como juegos que aceptan donaciones [21].

Encontramos dos opciones de compra para este articulo: la versión de 64GB por 349 euros y la versión de 256GB por 449 euros [22]. Ambas opciones incluyen el visor, dos controladores Touch, el cable del cargador, dos pilas AA, el adaptador de corriente y un separador para gafas.

Oculus Quest 2 dispone de otros accesorios que se venden por separado, como la correa para el visor antes comentada, un estuche para transportarlas (49 euros) [23] o el cable Oculus Link (99 euros) [24], que nos permite conectar las gafas al ordenador y ejecutar juegos y aplicaciones de Oculus Rift.

Estas gafas fueron lanzadas al mercado a finales del año pasado, con un precio muy atractivo en comparación con los otros dispositivos y un amplio catálogo de juegos y aplicaciones. Este ha superado el millón de unidades vendidas durante su primer trimestre, convirtiéndose así en el dispositivo de RV, no vinculado a un smartphone, que más ha vendido en los últimos meses [25].

Las grandes ventas de las Oculus Quest 2 ha sido el principal motivo para la elección de este para el desarrollo de nuestro videojuego.

La creciente comunidad de usuarios de las Oculus Quest 2 también ha supuesto un aumento de las impresiones 3D de accesorios para las mismas. En el caso de disponer de una impresora 3D, podremos imprimir algunos de los accesorios gratuitos que encontramos en Thingiverse [26].

En esta página podemos encontrar modelos 3D que han creado los usuarios para mejorar la comodidad de las gafas y evitar comprar la correa Elite que, como hemos indicado antes, cuesta 49 euros.

(a) Pieza para la parte posterior de la cabeza

(b) Sustitución de la correa por la de HTC Vive

Figura 2.19: Ejemplo de impresiones 3D para mejorar la correa de Oculus Quest 2

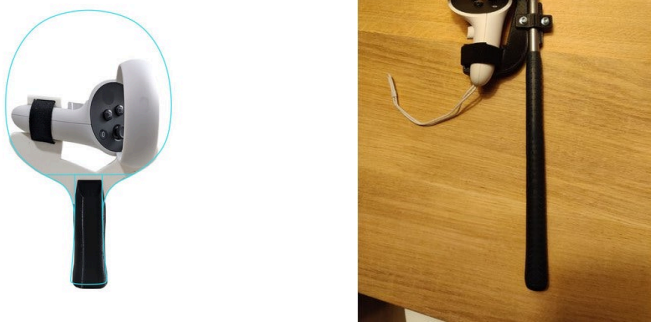
Además, encontramos otras impresiones que sirven para imitar la sujeción de un arma, como podemos ver en los siguientes ejemplos:

(a) Rifle

(b) Pistola

Figura 2.20: Ejemplo de impresiones 3D para imitar armas

También se pueden imprimir accesorios para deportes como por ejemplo el tenis de mesa o el golf:



(a) Pala de tenis de mesa **(b)** Palo de golf

Figura 2.21: Ejemplo de impresiones 3D para deportes

Podemos encontrar incluso impresiones de soportes para nuestras Oculus Quest 2 y muchas otras mejoras que los usuarios han diseñado.

(a) Soporte de mesa **(b)** Soporte de pared

Figura 2.22: Ejemplo de impresiones 3D para soporte de las Oculus Quest 2

En el caso de no disponer de una impresora 3D, podemos comprar impresiones 3D de accesorios para Oculus Quest 2 en Etsy [27]. Muchos de los accesorios que hemos visto gratuitos en Thingiverse están a la venta en Etsy, pero su precio puede ser bastante elevado. En el caso del rifle, este está a la venta por 48 euros.

2.3.1.5. Comparativa

En la Tabla 2.1 podemos encontrar una comparativa de las características técnicas más importantes de las gafas de RV que hemos mencionado anteriormente [28]:

DISPOSITIVO	PANTALLA	RESOLUCIÓN	TASA DE REFRESCO	CAMPO DE VISIÓN[3]	CONECTIVIDAD	PESO (g)	PRECIO (€)
Oculus Rift S	Single LCD	1280x1440	80 Hz	88° h, 88° v, 124° d	DisplayPort 1.2, USB 3.0	500	570
HTC Vive Pro	2 x AMOLED	1440x1600	90 Hz	88° h, 88° v, 124° d	HDMI, USB-C 3.0	550	1370
HTC Vive Cosmos	2 x LCD	1440x1700	90 Hz	99° h, 97° v, 139° d	HDMI, USB-C 3.0	702	879
HTC Vive Cosmos Elite	2 x LCD	1440x1700	90 Hz	99° h, 97° v, 139° d	HDMI, USB-C 3.0	702	999
Valve Index	2 x LCD	1440x1600	144 Hz	107° h, 104° v, 149° d	Front USB port (power only), DisplayPort 1.2, USB 3.0	809	1079
HP Reverb G2	2 x LCD	2160x2160	90 Hz	98° h, 90° v, 133° d	DisplayPort 1.3, USB 3.0	498	699
PlayStation VR	Single OLED	960x1080	120 Hz	96° h, 111° v, 147° d	HDMI, USB 3.0	600	300
Oculus Quest 2	Single Fast switch LCD	1832x1920	120 Hz Capada a 90 Hz	89° ho, 90° v, 127° d	USB-C, Oculus Link via USB-C, WiFi 6, Bluetooth 5.0 LE	503	349/449

Tabla 2.1: Comparativa de las características técnicas de los dispositivos de VR

[3]En la columna *CAMPO DE VISIÓN*, *h* hace referencia a horizontal, *v* a vertical y *d* a diagonal.

2.4. Videojuegos

En este apartado hablaremos en primer lugar de los videojuegos de VR mejor valorados, independientemente de su plataforma [29].

En primer lugar, debemos hablar de **Half-Life: Alyx**, el juego del visor Valve Index. Como hemos comentado antes, al comprar el visor se nos incluye este juego. Se trata de un *shooter* de acción y aventura en primera persona, en el que también encontramos fases de sigilo, puzles y una total interacción con los escenarios. Destaca por su enorme calidad en los gráficos y está disponible para Valve Index, HTC Vive, Oculus Rift, Windows Mixed Reality y Oculus Quest con ordenador y cable Link. Su precio es de 49,99 euros [30].

Figura 2.23: Half-Life: Alyx

Otro juego destacado de RV es **Beat Saber**, un videojuego de ritmo en el que deberás de cortar los cubos de colores que se dirigen hacia ti con sables de luz al ritmo de la música. Este cuenta con un modo multijugador. Se encuentra disponible para Valve Index, HTC Vive, Oculus Rift, Windows Mixed Reality y Oculus Quest. Su precio es de 29,99 euros [31].

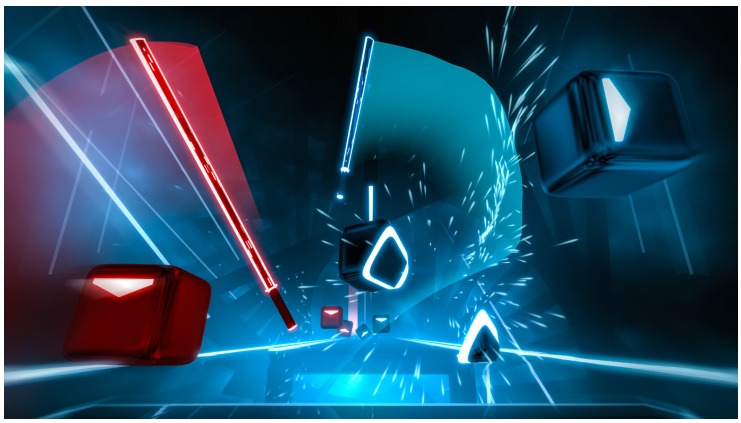


Figura 2.24: Beat Saber

Astro Bot Rescue Mission es un videojuego exclusivo para PlayStation VR. En este juego de aventura deberemos de acompañar a Astro para rescatar a la tripulación de bots de su nave espacial. Cuenta con 26 niveles en los que podremos encontrar puzles, plataformas y combates contra enemigos.Para poder jugarlo se requiere PlayStation VR, PlayStation Camera y el mando inalámbrico DUALSHOCK 4. Su precio es de 39,99 euros [32].

Figura 2.25: Astro Bot Rescue Mission

Otro videojuego destacado es **Tetris Effect**, una reformulación en 3D del clásico juego de Tetris. En este juego nos rodearemos de mundos tridimensionales que reaccionarán y evolucionarán en función de nuestra forma de jugar. Disponible para PlayStation VR, Oculus Rift, HTC Vive y Oculus Quest. Su precio es de 39,99 euros (29,99 euros para Oculus Quest) [33].

Figura 2.26: Tetris Effect

Por otro lado encontramos **SuperHOT**, un *shooter* donde el tiempo solo avanzará si te mueves [34]. En este no disponemos de una barra de salud que se regenera, sino que si te da una bala estás muerto. Podrás matar a los enemigos disparándoles, golpeándoles o lanzándoles objetos, y además podrás robarles sus armas. Estos podrán aparecer por cualquier lugar de la sala, por lo que deberás de estar atento y no descuidar ningún punto de la sala. Está disponible para Oculus Rift, Oculus Quest, HTC Vive, Windows Mixed Reality y PlaySation VR. Su precio es de 22,99 euros, menos en PlaySation VR donde es de 24,99 euros [35].

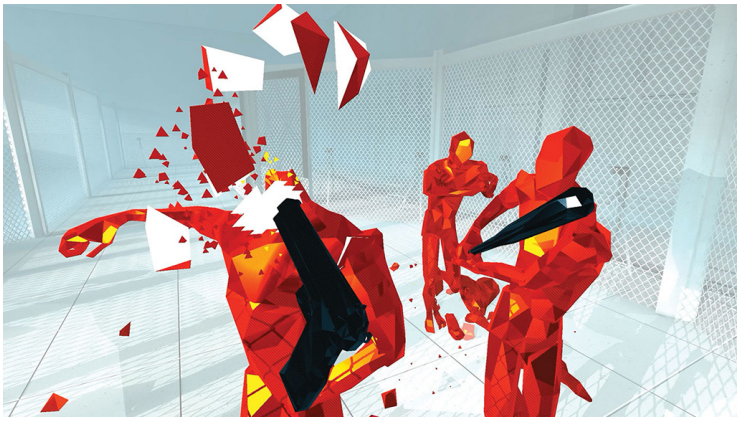


Figura 2.27: SuperHOT

Otro shooter popular en RV es **Arizona Sunshine**, donde deberás de sobrevivir a una Apocalipsis zombie en el Medio Oeste. Podrás explorar el mundo y matar a los zombies solo o en el modo multijugador. Un punto destacado del juego es que permite manejar más de 25

armas con movimientos de la vida real. Está disponible para Valve Index, HTC Vive, Oculus Rift, Windows Mixed Reality y PlayStation VR. El precio es de 29,99 euros para todos los dispositivos, con la excepción de nuevo de PlaySation VR que es de 39,99 euros [36].

Figura 2.28: Arizona Sunshine

Por último, para cambiar de género de juego hablaremos de **Trover Saves the Universe**. En esta cómica aventura deberás de ayudar a Trover a luchar contra Glorkon, un lunático que ha secuestrado a los perros y está utilizando la esencia de estos para destruir el universo. Podrás controlar los movimientos de Trover. Durante esta aventura nos encontraremos con combates, plataformas, acertijos y elecciones moralmente cuestionables. Disponible para Valve Index, HTC Vive, Oculus Rift, Oculus Quest y PlayStation VR. Su precio es de 24,99 euros para todos los dispositivos excepto Oculus Queste y PlayStation VR que es de 29,99 euros [37].

Figura 2.29: Trover Saves the Universe

2.4.1. Videojuegos en Oculus Quest

Debido a que nuestro videojuego se va a desarrollar empleando este visor, he creído conveniente mencionar otros videojuegos de RV centrándonos en los que están disponibles para Oculus Quest. Algunos de los videojuegos más destacados en este dispositivo los hemos comentado antes, como es el caso de **Beat Saber** o **SuperHOT**, pero podemos encontrar otros grandes videojuegos menos populares.

Este es el caso de **Eleven Table Tennis**, un simulador de tenis de mesa que destaca por su gran realismo. Podrás jugar en el modo multijugador online o practicar contra la IA. Su precio es de 19,99 euros [38].

Figura 2.30: Eleven Table Tennis

También encontramos **Sports Scramble**, un divertido juego en el que se mezclan varios deportes, de manera que podrás jugar al tenis con un palo de béisbol o jugar a los bolos con una pelota de rugby. Además, la bola podrá cambiar de manera que podrás estar jugando a tenis con una pelota de golf o una pelota de playa. Este juego cuenta con tres deportes principales: tenis, béisbol y bolos; y cuatro modos de juego: entrenamiento para un solo jugador, partida rápida, modos de desafío y el modo multijugador online. Su precio es de 29,99 euros [39].

Figura 2.31: Sports Scramble

Podemos encontrar otros buenos juegos de pago como es el caso de **The Climb** (29,99 euros), un juego en el que podrás experimentar la adrenalina de subir grandes alturas. Además, podrás disfrutar de increíbles paisajes de todo el mundo, como los Alpes, el sudeste asiático o el suroeste estadounidense. Podrás jugar solo o competir con tus amigos por obtener los mejores tiempos en el modo multijugador. Incluye también un modo turista donde se simplifican las mecánicas para facilitar su uso a los nuevos usuarios [40].

Figura 2.32: The Climb

Por otro lado, me gustaría destacar otros juegos menos populares en la tienda de Oculus Quest: **Hyper Dash** y **Cubism**. Estos juegos se dieron a conocer primero en la página *SideQuest* con una versión beta y una vez ganaron cierta popularidad pasaron a ser de pago en la página oficial de Oculus.

Hyper Dash es un *shooter* multijugador que cuenta con diversos modos de juego, como por ejemplo captura la bandera, combate a muerte o eliminación. Podrás comunicarte con los otros jugadores por medio del chat de voz. La única pega de este juego es que puede llegar a marear después de un par de partidas. Su precio actual es de 19,99 euros [41].

Figura 2.33: Hyper Dash

Cubism, sin embargo, es un juego de puzles en el que deberás de colocar correctamente las piezas para formar la figura 3D que se te indica. Cuenta con 60 puzles diferentes para poner a prueba tu ingenio. Su precio es de 9,99 euros [42].

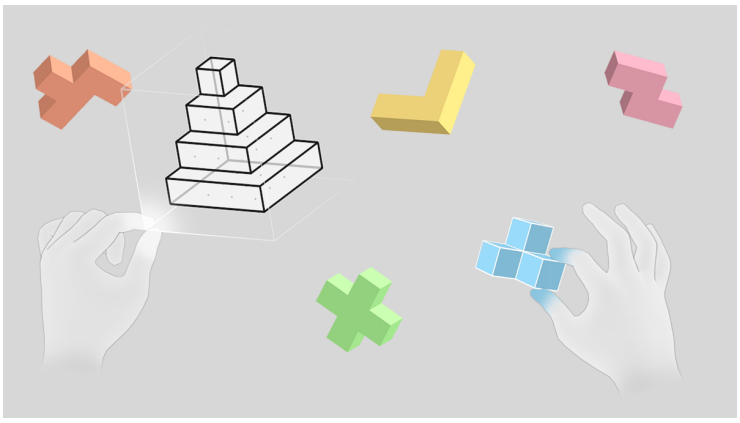


Figura 2.34: Cubism

Por último, destacar la beta gratuita del juego **Puzzling Places**. De nuevo se trata de un juego de puzles, pero, a diferencia del anterior, en este deberás de reconstruir miniaturas hiperrealistas de diversos lugares del mundo. Cada puzle se ha creado con un escaneo 3D

de fotogrametría[4]. Esta beta incluye seis puzles diferentes, entre los que nos encontramos el castillo de Tsuruga o el monasterio de Tatev [43].

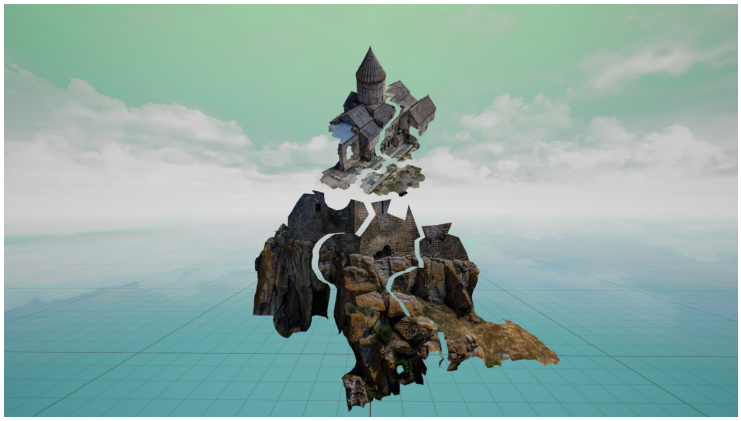


Figura 2.35: Puzzling Places

Cabe mencionar que en la página de *SideQuest* podremos encontrar muchos juegos, demos y versiones beta gratuitas. Estas dos últimas nos servirán para probar el juego y saber si lo compraremos cuando salga a la venta.

[4]Denominamos fotogrametría o fotometría al conjunto de técnicas y métodos que nos permiten componer una geometría tridimensional a partir de imágenes.

3. Metodología

Para el desarrollo de este proyecto emplearemos las metodologías ágiles. En concreto emplearemos Scrum, una de las metodologías ágiles más conocidas para la gestión de proyectos [44]. A pesar de que esta metodología está pensada para trabajar en equipos, muchas de sus prácticas nos serán muy útiles en el desarrollo de nuestro videojuego.

Scrum está especialmente indicado para proyectos donde se desee obtener resultados rápido y donde los requisitos son cambiantes o poco definidos. Esto hace que sea apropiado para nuestro proyecto, ya que necesitamos obtener un mínimo producto viable rápidamente para poder iterar sobre este. Además, al no haber desarrollado nunca un videojuego de este estilo, nuestros requisitos están poco definidos e irán cambiando a lo largo del desarrollo.

El proceso de desarrollo se dividirá en iteraciones, es decir, ciclos temporales cortos y de duración fija. A pesar de que estos normalmente suelen ser de 2-3 semanas, en nuestro caso este período será de un mes debido a que el desarrollo será llevado a cabo por una sola persona y que deberá compaginarlo con el curso académico. En cada iteración se priorizarán las tareas que hay que realizar en función del valor que aportan al proyecto en relación con su coste.

Una vez finalizada cada iteración, se procederá a realizar una revisión de la misma para ver los problemas que hayan podido surgir y replanificar el proyecto en caso de que fuera necesario.

Podemos agrupar las iteraciones en tres grandes fases:

- **Diseño e Investigación** Abarcará la primera iteración del proyecto. El objetivo de esta será definir el videojuego que queremos desarrollar e investigar las tecnologías existentes en el mercado. Para definir nuestro videojuego crearemos el documento de diseño del videojuego, de manera que podamos consultarlo en cualquier momento del desarrollo.

- **Desarrollo** Esta fase supondrá las cuatro siguientes iteraciones de nuestro proyecto. Durante esta fase realizaremos las pruebas necesarias para probar las diferentes tecnologías y procederemos al desarrollo de un mínimo producto viable. Además, deberemos de continuar con la escritura de la memoria.

- **Revisión y mejora gráfica** Esta última fase supondrá las dos últimas iteraciones del proyecto. El objetivo de esta será revisar el funcionamiento del videojuego y modelar los elementos que aparecerán en la escena. Además, se procederá a diseñar una nueva interfaz para el videojuego.

Para llevar un seguimiento de la planificación emplearemos la herramienta *Trello*. Esta nos servirá para poder ver a simple vista como va el proyecto y las tareas que quedan pendientes. Organizaremos las tareas en cuatro listas: *backlog*, donde tendremos todas las tareas necesarias para finalizar el proyecto; *to do*, donde pondremos las tareas a realizar en esa iteración; *in progress*, donde estarán las tareas que se están realizando en ese momento; y *done*, donde colocaremos las tareas finalizadas. Añadiremos una quinta lista donde pondremos los problemas que vayan surgiendo para no olvidarnos de estos.

Además, haremos uso de la herramienta de Toggl para llevar contabilizadas las horas que dedicamos a cada tarea, y GitHub para poder llevar un control de las versiones de nuestro proyecto.

3.1. Mínimo producto viable

El objetivo principal del desarrollo será alcanzar un mínimo producto viable para poder iterar a partir de él. Entenderemos como mínimo producto viable aquel videojuego en el que el jugador pueda seleccionar o crear una sala, conectarse a la misma, jugar una partida y ganarla o perderla. Con esta versión del producto pretendemos evitar que, debido a problemas internos o externos al desarrollo, alcancemos la fecha límite sin una versión jugable del mismo.

Una vez alcanzado ese objetivo, el siguiente paso será mejorar gráficamente el videojuego, agregando decorado a las escenas y mejorando las interfaces existentes. Posteriormente, en el caso de que de tiempo, procederemos a introducir efectos sonoros y animaciones a nuestro videojuego.

4. Documento de Diseño del Videojuego

4.1. Características

- **Título:** AirHockey VR
- **Plataforma:** Oculus Quest, Oculus Quest 2
- **Género:** Casual, Indie, Deporte, Simulación
- **Idioma:** Inglés
- **Público objetivo:** Cualquier persona mayor de 8 años
- **Modo de juego:** Multijugador

4.2. Descripción

AirHockey VR es un videojuego de Realidad Virtual en el que se simula el tradicional juego de hockey sobre mesa. En este juego dos personas compiten sobre una mesa de aire, en la que emplearán un *pusher*[1] para golpear el disco que deben meter en la portería contraria. Ganará, por tanto, el jugador que más goles haya anotado en el tiempo que dura la partida [45]. Se trata de un juego multijugador, por lo que el usuario deberá enfrentarse a otros jugadores reales.

4.3. Ambientación y estilo

Para este proyecto he optado por un estilo Low Poly[2] con texturas simples.

[1] Con pusher hacemos referencia al mazo que el jugador deberá coger y usar para golpear el disco

[2] El término Low Poly significa "bajo poligonaje". Es decir, es el modelado creado con el mínimo numero de polígonos posibles para un modelo 3D.

En el juego podemos encontrar dos escenarios distintos, uno para el *lobby* y otro para la partida. El primero se encuentra ambientado en una sala de espera, donde podremos encontrar dos puertas: una con el cartel de *exit* y otra con un cartel de *room*. De esta manera se pretende indicar al jugador las dos acciones que podrá hacer en esta habitación: entrar en partida o salirse del juego. Los distintos menús se muestran en una pantalla colocada en una de las paredes de la sala.

Por otro lado, al entrar en partida nos encontramos dentro de otra habitación que continuará con la estética del *lobby*, para así simular que hemos entrado a la puerta de *room* anteriormente comentada. Esta habitación está ambientada estilo *gamer*, con máquinas arcades y neones como se muestra en la figura 4.1. Este escenario será más oscuro que el anterior para imitar el ambiente de las salas recreativas.

Figura 4.1: Ejemplo de habitación Gamer

El centro de dicha habitación será nuestra mesa de air hockey, junto con los *pusher* y el

disco para jugar en la misma.

Por lo que respecta a los jugadores, ambos utilizarán el mismo modelo. Este únicamente cuenta con la cabeza y las manos del personaje. Con esto evitamos el problema que suele darse con los modelos de cuerpo entero en RV, ya que, al no tener piernas, estas no quedarán doblegadas cuando el jugador sea más pequeño que el modelo. Para crear este modelo nos inspiraremos en el casco que llevan los jugadores de hockey profesionales, que se muestra en la figura 4.2.

Figura 4.2: Casco de hockey profesional

4.4. Jugabilidad

4.4.1. Objetivo

Para ganar la partida, el jugador deberá marcar más goles que su oponente antes de que finalice el tiempo.

4.4.2. Mecánicas

El jugador podrá desplazarse por la habitación caminando. Además, podrá rotar por medio de los mandos.

De todos los objetos que encontramos en esta, el jugador únicamente podrá interactuar con los pushers y el disco. Cada jugador contará con su propio pusher, que empleará para golpear el disco y así marcar gol o evitar que le marquen.

4.5. Controles

En cuanto a los controles, debemos diferenciar entre los dos escenarios: el lobby y la partida.

Por lo que respecta al lobby, al encontrarnos a distancia, disponemos de un láser para indicarnos donde estamos apuntando. Para pulsar uno de los botones deberemos pulsar **A** en el mando derecho de las Oculus Quest, como podemos observar en la figura 4.3.

Figura 4.3: Mandos de Oculus Quest

Por otra parte, los controles durante la partida son distintos. Para girarnos emplearemos el **Thumb Sticks** derecho, que podemos ver señalado en la figura 4.4. Para coger el pusher o el disco será necesario pulsar el botón de **Grip**. Por último, para pulsar el botón de volver al lobby nos aparecerá de nuevo el láser anteriormente comentado.

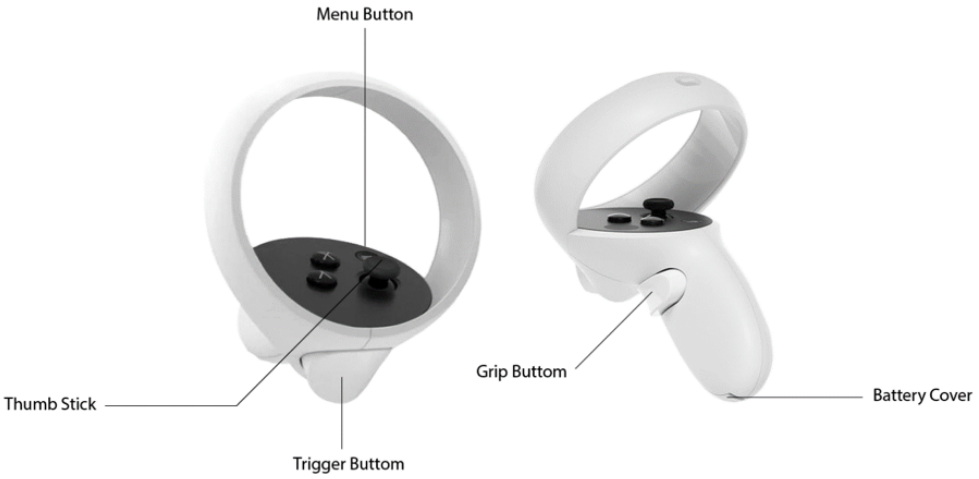


Figura 4.4: Mandos de Oculus Quest 2 con los botones identificados

El jugador podrá cerrar el puño al completo pulsando el botón de **Grip** o juntar el dedo índice y pulgar pulsando el botón de **Trigger**.

En cualquier momento podremos salir del juego pulsando en el botón de Menú del mando derecho.

4.6. Pantallas

En esta sección se mostrarán una serie de mockups que hacen referencia a las distintas pantallas que podremos encontrar en el juego.

En primer lugar encontramos la pantalla para introducir el nombre. Esta pantalla y las tres siguientes hacen referencia al lobby y por tanto aparecerán integradas en la pared de la sala como se ha comentado en la sección anterior. En esta pantalla en concreto aparecerá un teclado para que el usuario pueda introducir su nombre.

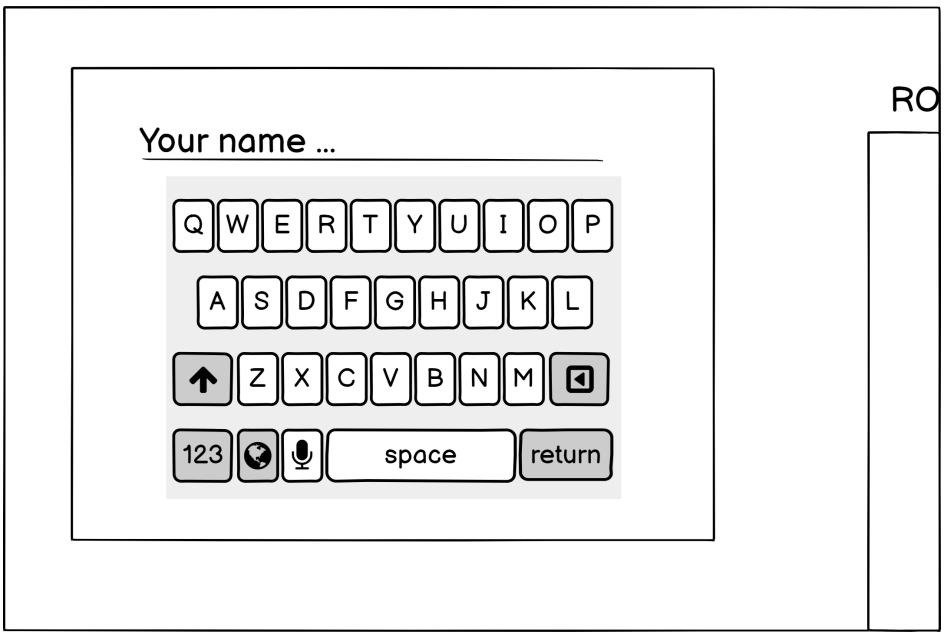


Figura 4.5: Pantalla para introducir nombre

Una vez introducido el nombre, el usuario pasará a ver una lista con todas las salas disponibles. Además, dispondrá de un botón para recargar la lista de salas y otro para crear una propia.

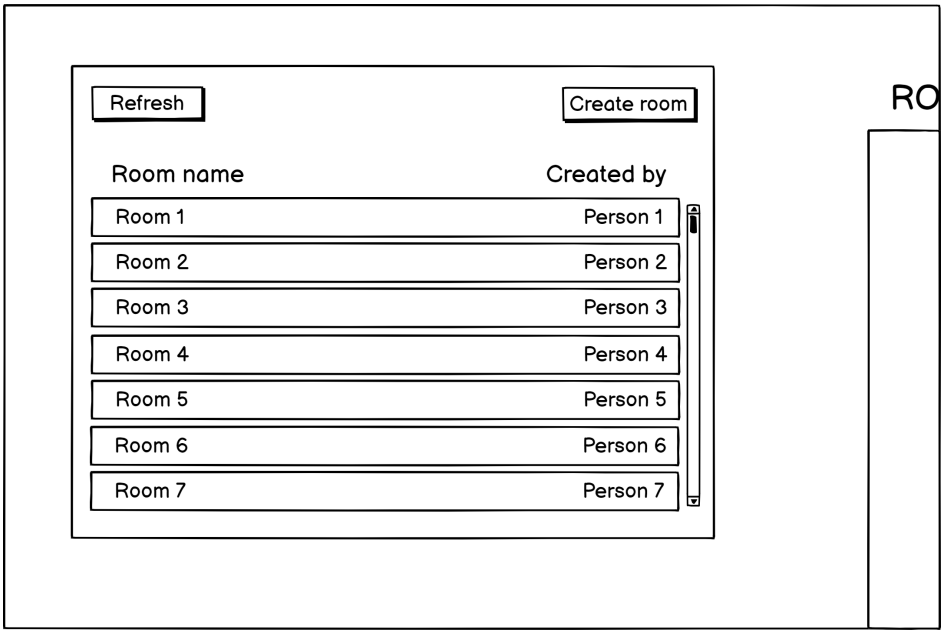


Figura 4.6: Pantalla con la lista de salas

Si el jugador selecciona crear su propia sala, se le mostrará la siguiente pantalla. Esta cuenta de nuevo con un teclado para introducir el nombre de la sala y con un botón para volver a la pantalla anterior.

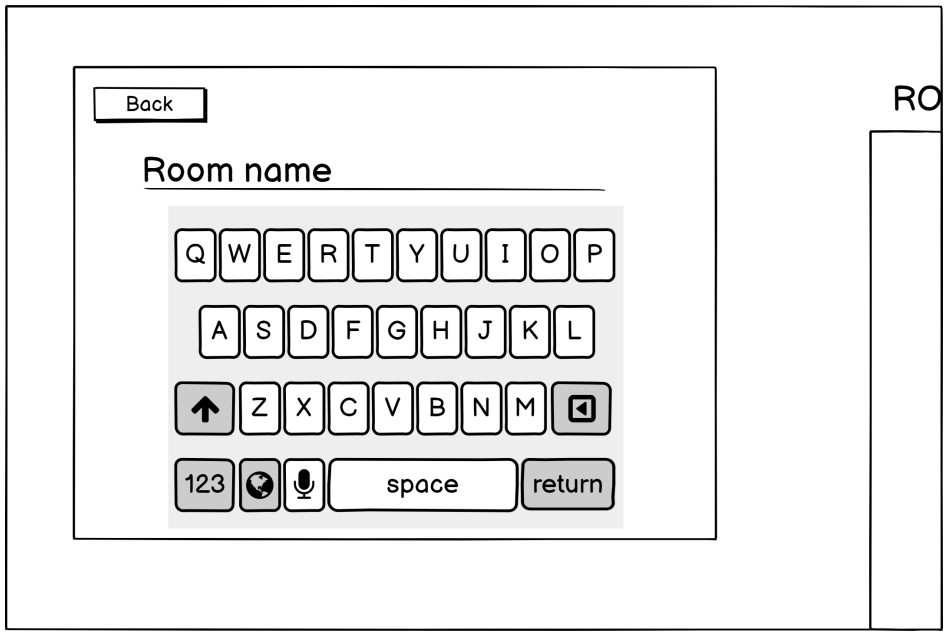


Figura 4.7: Pantalla para crear sala

Al crear una sala, al jugador se le mostrará la siguiente página de espera hasta que otro jugador seleccione su sala.

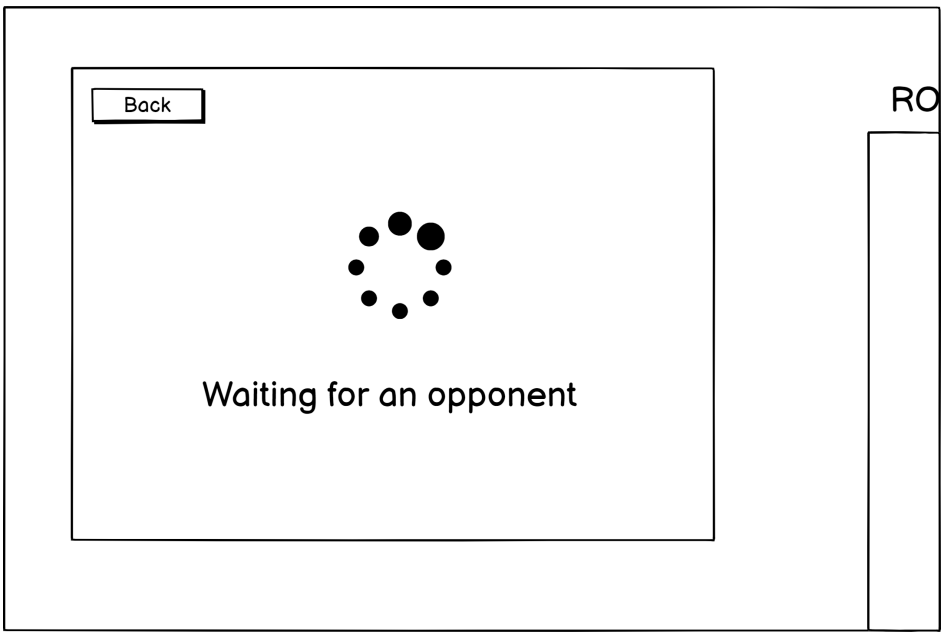


Figura 4.8: Pantalla de espera después de crear la sala

Una vez dentro de la partida, lo primero que se encontrará el jugador es una cuenta atrás para iniciar la partida.

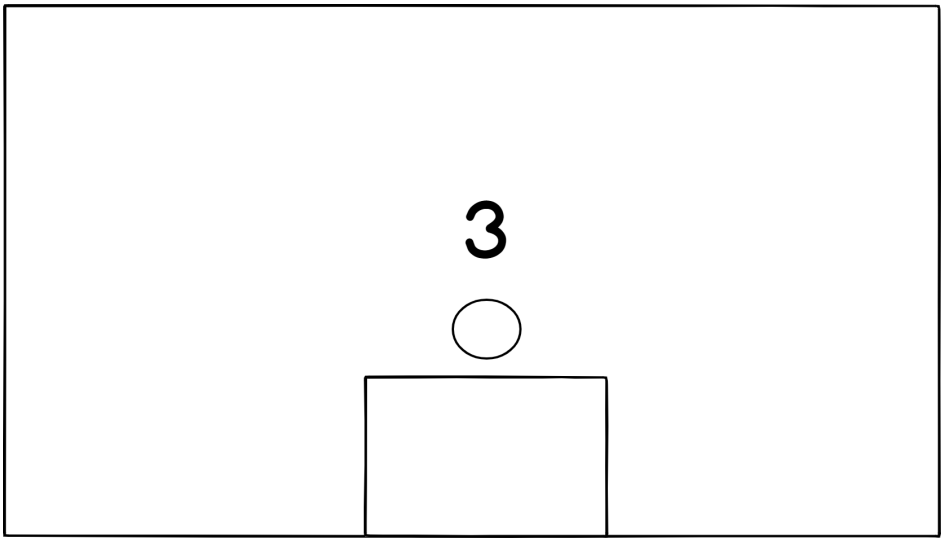


Figura 4.9: Cuenta atrás para comenzar la partida

Una vez comenzada la partida, el jugador podrá ver el tiempo, el marcador y el nombre del otro jugador, además del resto de objetos presentes en la habitación y que se han comentado

en la sección de ambientación.

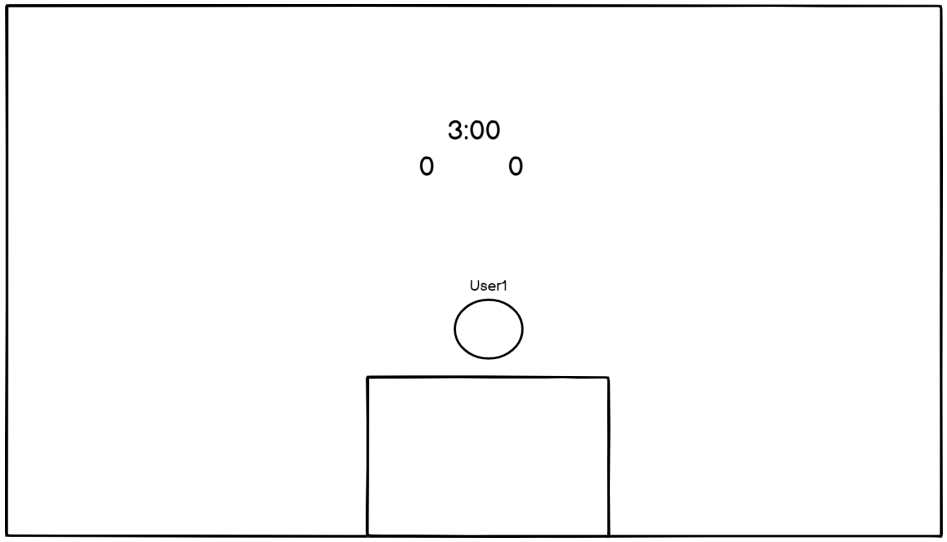


Figura 4.10: Durante la partida

Una vez finalizado el tiempo, al jugador se le mostrará un texto indicándole si ha ganado o ha perdido y un botón para volver a la pantalla con la lista de salas (lobby).

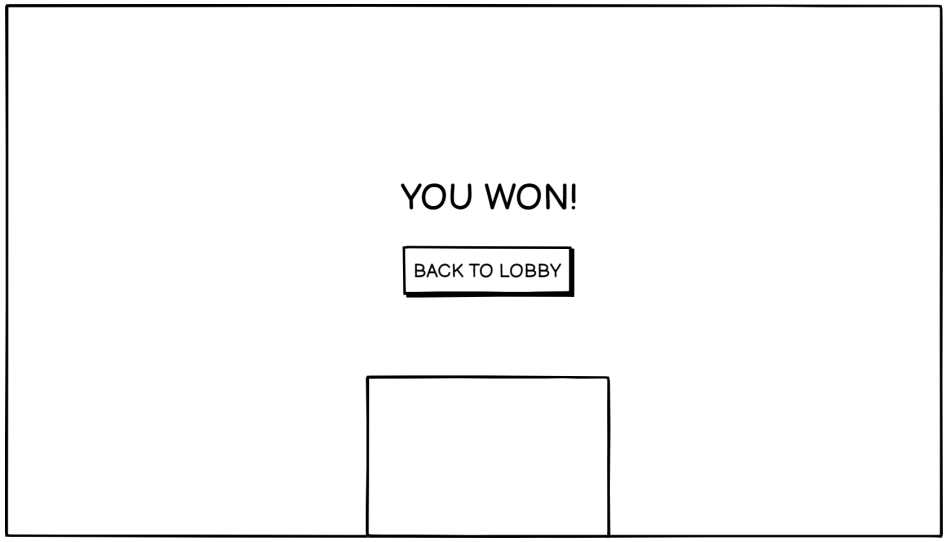


Figura 4.11: Fin de la partida

4.7. Estados de Juego

A continuación se muestra un diagrama sobre la relación entre pantallas arriba comentada. Con este diagrama será más fácil identificar a simple vista el flujo entre estados a la hora de programar.

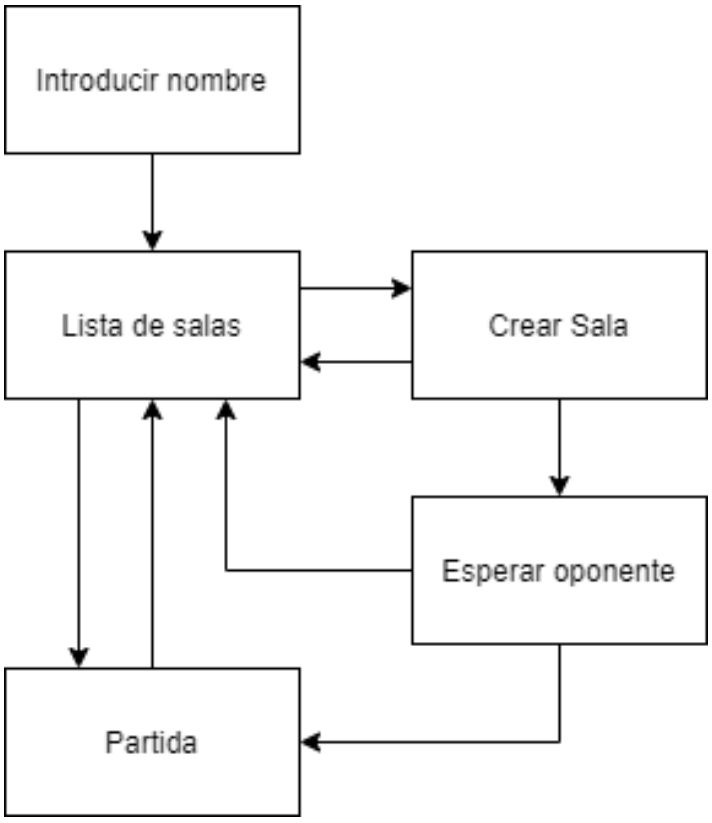


Figura 4.12: Diagrama de flujo

Al iniciar el juego, el jugador deberá introducir su nombre para poder acceder a una sala. Una vez introducido podrá seleccionar una sala de la lista o podrá crear una propia. En caso de decantarse por esta última opción deberá esperar a que otro jugador seleccione su sala para empezar la partida. Una vez en partida, para volver a la lista de salas será necesario que finalice el tiempo de partida.

4.8. Sistema sonoro

4.8.1. Música

Por lo que respecta a la música ambiente, emplearemos dos estilos de música distintos para diferencias ambos escenarios.

Para el *lobby* seleccionaremos una canción del género Lo-fi [3].La selección de este género se debe a sus ritmos tranquilos, que nos permitirán evitar que el jugador se canse de escucharla mientras espera.

Por otro lado, para la partida deberemos de seleccionar una canción enérgica que combine adecuadamente con el ritmo de juego.

4.8.2. Efectos de sonido

Será necesario introducir efectos de sonido durante la partida. Entre estos efectos debemos destacar el sonido del golpe entre el disco y el *pusher*, el disco con los márgenes de la mesa o el sonido del disco al caer a la zona de reaparición cuando marcamos gol. Además, deberemos agregar un efecto de sonido para indicar que se ha marcado gol.

[3]El lo-fi (abreviado del inglés low fidelity) es un enfoque de producción en el que predomina el uso de medios anticuados o de baja fidelidad de grabación, a veces como una mera decisión estética.

5. Herramientas

En esta sección hablaremos de las herramientas empleadas para el desarrollo del videojuego. Además, analizaremos las diferentes opciones para el motor de juego y el sistema de multijugador y seleccionaremos la que mejor se adapte a nuestro proyecto.

5.1. Hardware

5.1.1. Oculus Quest 2

El Oculus Quest 2 es el visor de RV elegido para nuestro proyecto. Cabe destacar que tanto el Oculus Quest como el Oculus Quest 2 son válidos para el desarrollo y en ambos se podrá ejecutar el juego.

Este visor ha sido desarrollado por Oculus, una sección de la empresa Facebook. El primer visor, Oculus Quest, fue lanzado al mercado en mayo de 2019 [46]. En octubre de 2020 sacaron el Oculus Quest 2, que tiene un diseño similar pero con menos peso, una pantalla con mayor tasa de refresco y mejoras en las especificaciones internas y en los controladores Oculus Touch. Ambos visores cuentan con un sistema operativo basado en Android [47].

La elección de dicho visor se debe a la libertad de movimiento que proporciona al usuario, ya que al disponer de un procesador integrado no es necesario estar conectado por cable al ordenador. Esto permite al usuario desplazarse por la estancia sin ningún problema.

Dicha elección se fundamenta también en el aumento de usuarios que compran este visor. Gracias a su económico precio[1], se sitúan como una de las mejores opciones para RV, estableciendo un nuevo récord con más de 1 millón de dispositivos vendidos en su primer trimestre. Y es por ello que cada vez es mayor el número de desarrolladores de videojuegos para este

[1] El precio de las Oculus Quest 2 es de 349€ su versión básica de 64GB de almacenamiento interno, pudiendo ser ampliado a 256GB por 100€ más.

dispositivo.

5.2. Software

5.2.1. Motor de juego

A la hora de seleccionar el motor más apropiado para nuestro videojuego nos centramos en los motores más demandados por la industria actual y descartamos otros menos populares. Es por ello que nos centramos en Unity y en Unreal Engine [48].

En la actualidad, estos son los motores más utilizados en los estudios de desarrollo de videojuegos, y suele pedirse cierta experiencia manejando uno de estos motores para optar a un puesto de trabajo, incluso en empresas que utilizan un motor propio.

Ambos son motores gratuitos, muy potentes y con una gran comunidad detrás. No obstante, cabe decir que debido a las nuevas actualizaciones y al hecho de que cada vez sean más accesibles, cada vez son más las personas que crean videojuegos como pasatiempo y no de forma profesional.

5.2.1.1. Unity

Unity es un motor de juego creado por Unity Technologies. La primera versión de este se lanzó en 2005 y su objetivo era proporcionar herramientas accesibles para equipos independientes y de tamaño reducido [49].

Es uno de los motores más empleados para el desarrollo de videojuegos, como se puede observar en sus cifras del periodo de 2020 [50]:

- **5B** de descargas al mes de aplicaciones creadas con Unity
- **71%** de los 1000 juegos móviles principales se creó con Unity
- **Más del 50%** de los juegos para dispositivos móviles, PC y consolas se creó con Unity
- **2.5B** de usuarios activos mensuales que consumieron contenido creado u operado con soluciones de Unity
- **Más de 20** plataformas diferentes ejecutan proyectos creados con Unity

- **Más de 190** países y territorios tienen creadores de Unity

5.2.1.2. Unreal Engine

Unreal Engine es un motor de juego creado por la compañía Epic Games. Su primera aparición data de 1998, pero la primera versión gratuita no se publicó hasta 2009 con Unreal Engine 3: El Unreal Development Kit para permitir a grupos de desarrolladores amateur realizar juegos con el Unreal Engine 3 [51].

No obstante, el gran salto de dicho motor se produjo con la cuarta versión de este en 2015. Los avances en esta última versión le han convertido en el principal competidor de Unity.

5.2.1.3. Comparativa

Para decantarnos entre ambos motores había que tener en cuenta diversos aspectos, que desarrollaremos a continuación [52]:

- **Tienda de assets:** ambos motores cuentan con una amplia gama de productos en sus tiendas, no obstante la tienda de Unity es cinco veces mayor.
- **Documentación y formación:** tanto Unity como Unreal tienen una extensa documentación acerca del motor y cómo usarlo. No obstante, el número de cursos y tutoriales para Unity es mucho mayor.
- **Lenguaje de programación:** en Unity el lenguaje principal es C#, mientras que en Unreal puedes optar entre programar con C++ o emplear Blueprints, que es un sistema de scripting visual. C# destaca por ser más fácil que C++, pero con este último se pueden desarrollar juegos más optimizados. Cabe destacar también que existe la posibilidad de integrar el scripting visual en Unity gracias a un plugin denominado Bolt [53].
- **Comunidad de desarrolladores:** de nuevo es mucho mayor en el caso de Unity, por lo que en el caso de encontrarnos con alguna dificultad será más probable que encontremos a otra persona con el mismo problema. Además, cabe destacar que la comunidad de Unreal se divide entre programadores y no programadores que emplean los Blueprints, siendo estos últimos el doble que los primeros.

- **Gráficos:** se pueden producir resultados similares con cualquiera de los dos motores. Sin embargo, destaca Unreal por sus mayores capacidades [54].

- **Cuerva de aprendizaje:** esta es mucho menor en Unity. Además, este cuenta con una interfaz mucho más intuitiva y simple que Unreal [55].

- **Precio:** ambos motores son gratuitos. No obstante, por lo que respecta a Unreal, si tu juego produce más de 3000 dólares, debes pagar un 5% de tus ganancias totales. En cuanto a Unity, si las ganancias de tu juego superan los 100.000 dólares al año, tienes que pagar una mensualidad de 40 dólares al mes.

Como podemos observar, ambos motores son una buena opción y nuestra elección se basará en las características del juego a desarrollar.

En nuestro caso particular, se trata de un videojuego de RV con gráficos low poly, por lo que este último aspecto no será relevante a la hora de tomar una decisión. Por otro lado, la tienda y la comunidad de desarrolladores de Unity hace que nos decantemos por este, ya que nos puede resultar de gran utilidad. El único aspecto en el que vence Unreal es en el hecho de programarse en C++.

No obstante, debido a las ventajas que supone la gran comunidad de Unity y el amplio número de tutoriales, este motor será más adecuado para iniciarse en el mundo de la RV.

5.2.2. Oculus Integration

Unity provee de un soporte integrado de RV para los dispositivos de Oculus. El paquete de Oculus Integration únicamente agrega *scripts*, *prefabs*, ejemplos y otros recursos a dicho soporte de Unity [56].

Este paquete es gratuito y lo podemos encontrar fácilmente en la *Asset Store* de Unity. Entre todas las funcionalidades que incluye el paquete, cabe destacar las siguientes [57]:

- Una interfaz para controlar el comportamiento de la cámara de RV.

- Un *prefab* que permite al jugador moverse por el entorno virtual.

- Una API de *input* unificada para los controladores.

- Un *script* que permite coger y soltar objetos usando los controladores Oculus Touch.
- Un *script* que permite a los objetos ser cogidos y soltados con los controladores de Oculus.
- Herramientas de *debug*.

5.2.3. Sistema de multijugador

Una vez decidido el motor de juego a emplear, el siguiente paso es elegir el sistema de multijugador que vamos a utilizar. Para ello analizaremos brevemente los posibles candidatos y seleccionaremos aquel que mejor se adapte a nuestras necesidades.

5.2.3.1. UNet

Es la herramienta proporcionada por Unity para realizar juegos multijugador. No obstante, como se indica en su página, este está deprecado y será eliminado de Unity en un futuro. Informan también de que el nuevo sistema que sustituirá UNet está en fase de desarrollo actualmente [58].

El hecho de aprender una herramienta que va a desaparecer y el tener que pasarlo al nuevo sistema de multijugador de Unity cuando se lance al mercado, hace de este una mala opción para nuestro proyecto.

5.2.3.2. Photon PUN 2

Photon es un framework de desarrollo de juegos multijugador en tiempo real. Este destaca por ser rápido, sencillo y flexible. Photon consta de un servidor y varios SDK de cliente para las principales plataformas.

Photon Unity Network (PUN) es la versión específica de Unity. Este destaca por la sencillez de su API, la extensa documentación y tutoriales de cómo emplear el mismo, los servidores dedicados y la posibilidad de exportarlo a todas las plataformas (móvil, consolas, aplicaciones de escritorio, TV, VR, AR y web). Todas las aplicaciones se ejecutan en Photon Cloud. Son ellos los que se encargan de los servicios de alojamiento, operaciones y escalado [59].

PUN cuenta con una versión gratuita de 20 CCU y 60.0GB, que indican que es perfecta para las etapas de desarrollo y evaluación. Sin embargo, si queremos mejorar este, deberemos pagar 95 dólares al mes para obtener 500 CCU y 1.5TB o precios más elevados para unas mejores prestaciones.

Además, Photon cuenta con un sistema de voz gratuito que es fácil de introducir en nuestro proyecto y que permitiría la comunicación entre ambos jugadores durante la partida.

Podemos encontrar diversos tutoriales de cómo emplear este en juegos de RV, tanto de los propios creadores del framework como de la gran comunidad de desarrolladores que encontramos detrás. Además, en la propia documentación encontramos muchos tutoriales y demos acerca de temas importantes como son el lobby, el matchmaking y la compensación del lag. PUN, por tanto, permite de manera sencilla la gestión de salas y la limitación del número de usuarios en las mismas y la creación de un lobby en el que mostrar las salas disponibles.

Por otro lado, PUN asegura una latencia global baja debido a la existencia de centros de hosting de Photon Cloud repartidos por todo el mundo. Otro aspecto positivo de PUN es el hecho de que es compatible también con Unreal Engine.

Photon cuenta también con otros productos como Realtime, Bolt y Quantum. No obstante, la documentación y tutoriales de cómo emplear estos para juegos de RV son escasos, por lo que han sido descartados de esta comparativa.

5.2.3.3. Normcore

Normcore es un framework desarrollado por la empresa Normal. Este surgió mucho después que Photon PUN, por lo que la comunidad de desarrolladores es mucho menor y es más difícil encontrar tutoriales de este para RV. No obstante, la comunidad de Normcore en Discord es bastante activa y los propios desarrolladores están muy pendientes de las preguntas que le escriben por los canales de texto.

Al igual que con el anterior, podemos encontrar muchos aspectos positivos en el uso de este sistema de multijugador [60]:

- **Sincronización Automática:** únicamente es necesario agregar el componente RealtimeTransform a cualquier objeto de la escena para obtener la sincronización automática de la transformación de dicho objeto. No es necesario código.

- **Física:** interpolación de última generación y física en red robusta.
- **Espacios persistentes:** cualquier objeto multijugador en Normcore puede persistir entre sesiones con un solo clic o una sola línea de código.
- **Chat de Voz:** proporciona capacidades de transmisión de audio y VOIP estándard en la industria. Para incorporarlo en nuestro proyecto únicamente necesitaremos incluir un componente de Unity.
- **Compatible con XR:** Normcore es compatible con XR de fábrica. Puedes crear aplicaciones que funcionen en todos los dispositivos de RV y RA sin ningún cambio en su código.
- **Extensible:** Se pueden crear fácilmente componentes personalizados para sincronizar cualquier dato en el proyecto.

De nuevo, cuenta con una API sencilla e intutiva y tiene soporte para todas las plataformas nativas (Windows, Mac, iOS, Android, Playstation, Xbox, Nintendo Switch, Oculus Quest, etc.). Cuenta además con detección automática de región, de manera que ejecutará los servidores de sala en la región ideal para cada cliente. Al igual que el anterior, los servidores de Normcore se ejecutan en regiones de todo el mundo y están conectados a través de una res de fibra privada, lo que proporciona una latencia baja.

De nuevo encontramos una versión gratuita limitada a 30 usuarios concurrentes, 10 salas, límite de sesión de 1 hora, 50 horas de habitación y 120GB de ancho de banda. Para mejorar estos aspectos será necesario pagar 49 dólares al mes para la versión PRO o 49 dólares al mes más uso para la versión Ilimitada.

A diferencia de Photon PUN, Normcore no está disponible para Unreal Engine.

5.2.3.4. Comparativa

Después de investigar las características de cada sistema, el siguiente paso es implementar una escena básica en Unity para probar el funcionamiento de ambos. Esta escena únicamente contendrá un cubo que simula una mesa y otros cubos más pequeños para poder interactuar con ellos.

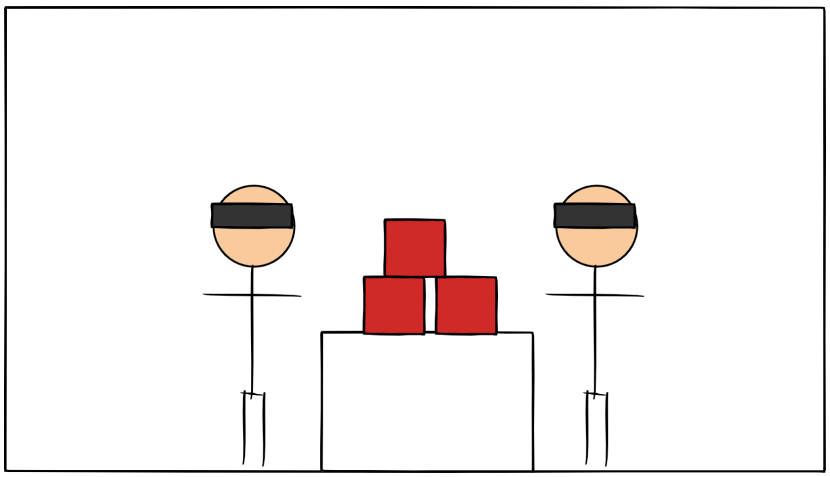


Figura 5.1: Ejemplo de la escena de prueba con dos usuarios

El objetivo de dicha prueba es poder comparar el funcionamiento a tiempo real de ambos sistemas y seleccionar el que mejor se adapte a nuestro proyecto. Para poder probarlo correctamente necesitaremos dos personas con Oculus Quest que accedan a la sala y que interactúen con los cubos.

Después de realizar la prueba, podemos confirmar que el mejor sistema para nuestro proyecto es Normcore, ya que la latencia es mucho menor y la interacción entre dos objetos funciona mejor en este. Es decir, al golpear un objeto con otro estos no se mueven apropiadamente en Photon PUN 2 y esto supone un grabe problema, ya que nuestro juego se basa en la interacción entre disco y *pusher*. Esta decisión supone que la gestión de salas será responsabilidad nuestra, ya que este no nos proporciona ninguna herramienta para ello.

5.2.4. MongoDB Atlas

MongoDB Atlas es un servicio de base de datos en la nube [61]. Es, por tanto, una forma de emplear MongoDB en la nube. MongoDB es un sistema de base de datos NoSQL, orientado a documentos y de código abierto. Esto implica que en lugar de almacenar los datos en tablas, como en las bases de datos relacionales, MongoDB almacena las estructuras de datos en documentos BSON con un esquema dinámico y estos se almacenan a su vez en colecciones. Podríamos decir que una colección equivale a una tabla en las bases de datos relacionales y un documento sería como un registro en dicha tabla. A diferencia de las relacionales, en

MongoDB los documentos de una misma colección pueden contener campos diferentes [62].

Emplearemos dicha base de datos para gestionar las salas creadas para el multijugador, ya que, como hemos indicado arriba, Normcore no nos ofrece ninguna herramienta para ello.

5.2.5. Blender

Blender es un programa gratuito y de código abierto para la creación 3D. Entre sus funcionalidades podemos encontrar el renderizado, modelado, esculpido, la animación y *rigging*, los VFX y la edición de vídeo [63].

En nuestro proyecto lo hemos utilizado para el modelado de los elementos 3D que podemos encontrar en el videojuego.

5.2.6. GIMP

GIMP es un programa de edición de imágenes digitales en forma de mapa de bits. Este programa es libre y gratuito [64].

En este videojuego lo hemos empleado principalmente para la creación de texturas de los modelos 3D.

5.2.7. Inkscape

Inkscape es un programa de edición de gráficos vectoriales. Este es gratuito y de código abierto. Trabaja principalmente con el formato vectorial escalable (SVG) [65].

En este TFG lo hemos utilizado para la creación de los elementos gráficos de la interfaz de nuestro videojuego.

5.2.8. Audacity

Audacity es un programa de edición de audio. Al igual que el anterior, este es gratuito y de código abierto. Permite tanto la grabación de audio como el procesamiento posterior [66].

Lo emplearemos para recortar y reducir el ruido de los efectos de sonido de nuestro videojuego.

6. Desarrollo

En este apartado desarrollaremos los distintos pasos realizados para la implementación del videojuego. Incluiremos también la explicación de las pruebas realizadas para la selección del sistema de multijugador.

6.1. Configuración del proyecto

Para poder comenzar con el desarrollo de nuestro videojuego, el primer paso a realizar será la configuración del proyecto de Unity para que se pueda ejecutar en el Oculus Quest 2 [67].

En primer lugar deberemos instalar la última versión estable de Unity, en el caso de que no lo tengamos ya instalado. En mi caso se trata de la versión 2019.4.17f1. Durante la instalación, deberemos de asegurarnos de marcar el módulo de Android como se muestra en la figura 6.1, ya que, como hemos mencionado anteriormente, el Oculus Quest cuanta con un sistema operativo basado en este.

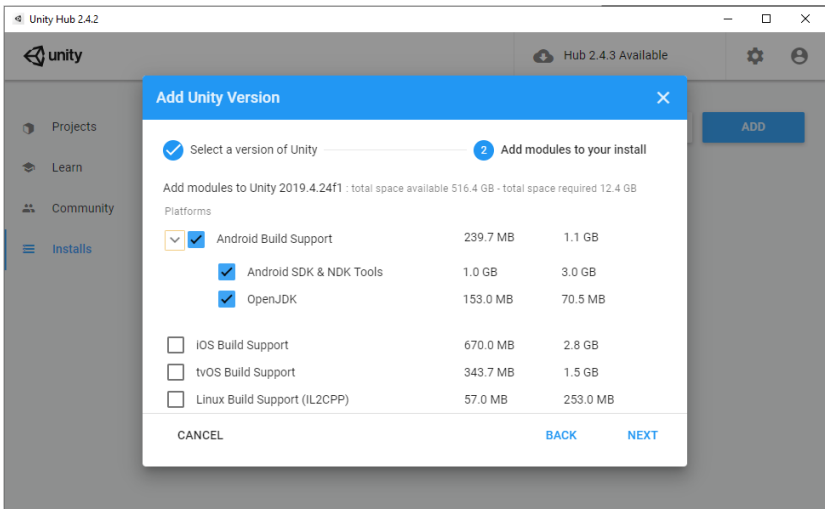


Figura 6.1: Módulos necesarios para compilar para el Oculus Quest

Una vez lo tengamos instalado, deberemos crear un nuevo proyecto y cambiar en este el sistema operativo para el que vamos a compilar. Para ello tenemos que abrir la pestaña *Build Setting*, seleccionar Android y pulsar al botón de *Switch Platform* que se muestra en la figura 6.2.

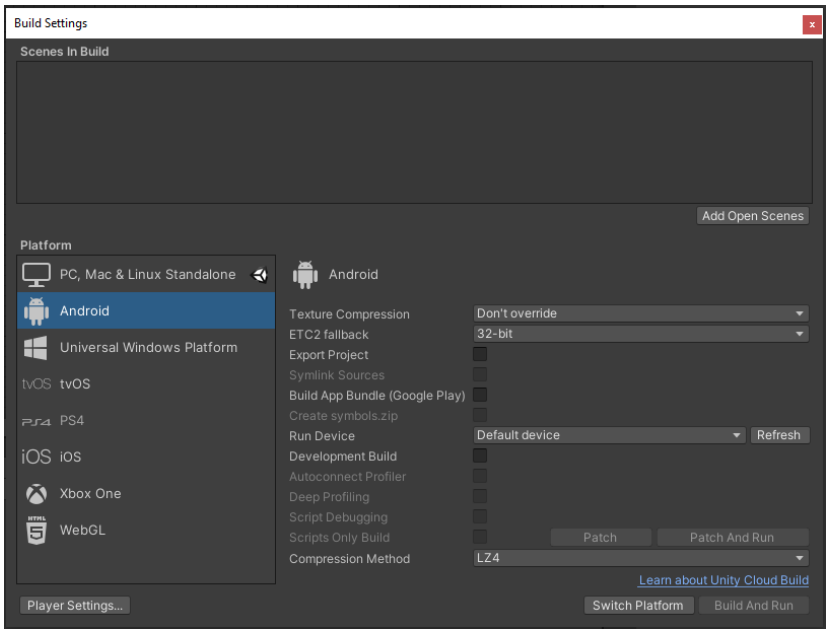


Figura 6.2: Pestaña para configurar el sistema operativo para el que vamos a compilar

El siguiente paso será importar el paquete de Oculus Integration. Para ello deberemos acceder al *Asset Store* desde la pestaña *Window* del menú y buscar en ella Oculus Integration. Una vez encontrado el paquete, deberemos descargarlo y a continuación pulsar en el botón de *Import* que podemos ver en la figura 6.3.

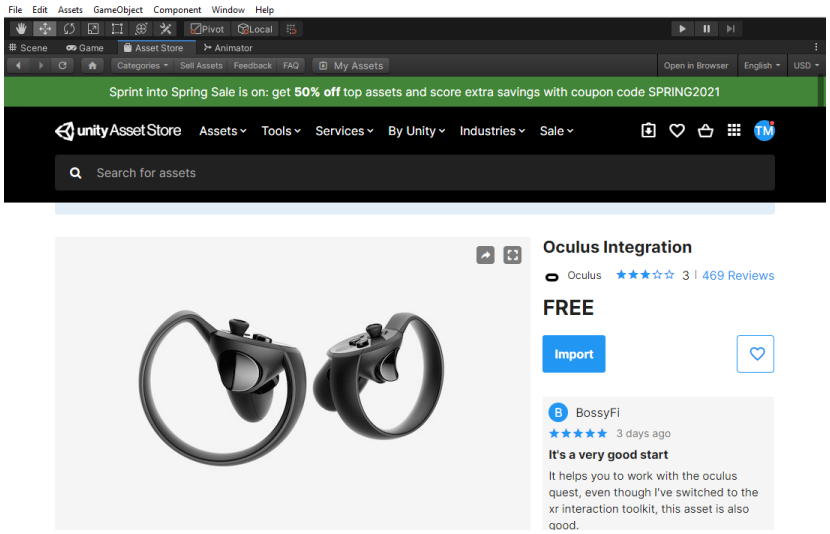


Figura 6.3: Asset Store de Unity una vez descargado el paquete Oculus Integration

A continuación debemos habilitar el soporte de RV en Unity. Para ello abriremos la pestaña de *Project Settings* y en el panel izquierdo seleccionaremos el apartado de *XR Plugin Management*. Seguidamente pulsaremos sobre el botón de *Install XR Plugin Management*. Una vez instalado, solo nos queda seleccionar Oculus en la lista que nos aparecerá para instalar el *plugin* de Oculus XR, como se observa en la figura 6.4.

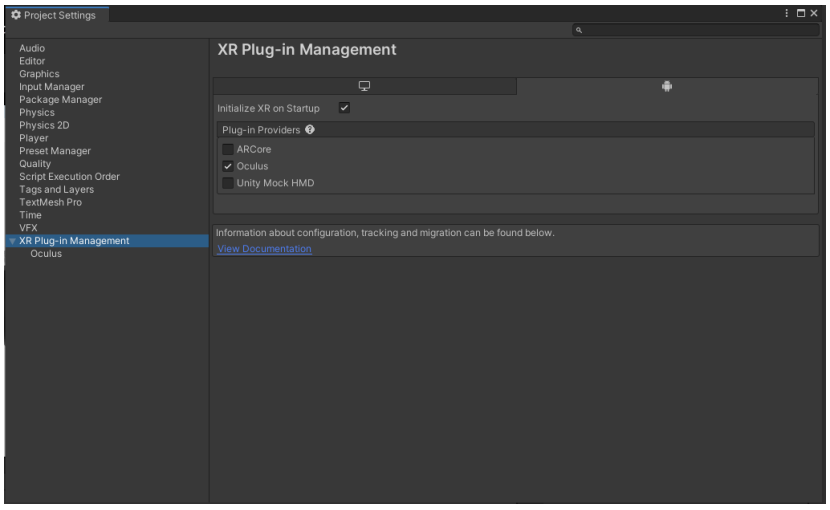


Figura 6.4: Ventana de XR Plug-in Management con la opción de Oculus seleccionada

Una vez tenemos todo instalado, el siguiente paso es eliminar la cámara que nos viene por

defecto en el proyecto y sustituirla por el *prefab* denominado OVRPlayerController, que nos lo proporciona el paquete de Oculus Integration. Este *prefab* permite al jugador desplazarse por la escena e incluye otro *prefab*, OVRCameraRig, que sirve cómo camara de RV. Esta cámara se moverá siguiendo los movimientos del visor.

El último paso para terminar esta configuración será proporcionarle al jugador la capacidad de coger y mover objetos. Para ello, incorporaremos el *prefab* LocalAvatar dentro del OVRPlayerController de la escena, más concretamente como hijo del objeto TrackingSpace. Este *prefab* proporciona al jugador unas manos virtuales dentro de la escena. Para poder interactuar con los objetos será necesario agregar a los objetos LeftHandAnchor y RightHandAnchor, hijos del objeto TrackingSpace, un *Rigidbody*, un *sphere collider* que debe ser *trigger* y el *script* de OVRGrabber, tal y como se muestra en la figura 6.5.

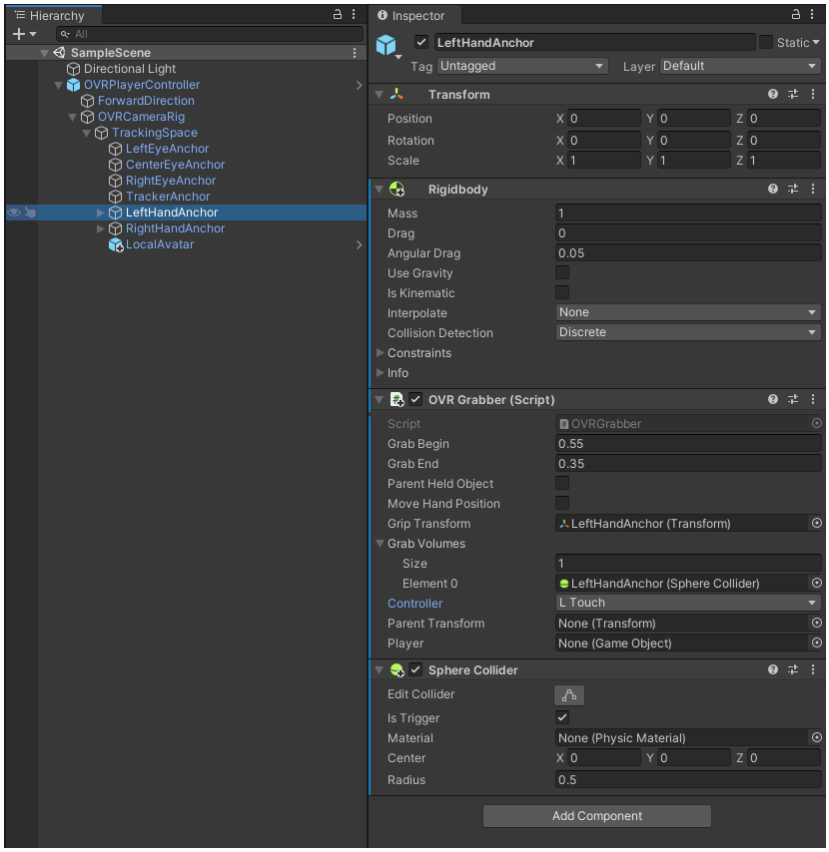


Figura 6.5: configuración de OVRPlayerController para poder coger y mover objetos

6.2. Pruebas de los sistemas de multijugador

Para esta prueba prepararemos en primer lugar un escenario base para emplear en ambas pruebas. Este escenario estará compuesto por un suelo, un cubo grande que simulará una mesa y varios cubos pequeños para interactuar con ellos.

Para ambas pruebas será necesario haber configurado previamente el proyecto para RV.

6.2.1. Photon PUN 2

El primer paso para emplear este sistema de multijugador es descargarlo e importarlo al proyecto desde la *Asset Store* de Unity. A continuación, deberemos crearnos una cuenta en la página web de Photon. Una vez registrados, deberemos de crear una nueva aplicación, cuyo id emplearemos para configurar el proyecto en Unity.

Por tanto, para configurar Photon en el proyecto deberemos de acceder a *Window > Photon Unity Networking > Highlight Server Settings* e introducir el id de la aplicación que hemos creado antes en el apartado de *App Id Realtime*.

El siguiente paso será conectarnos al servidor. Para ello crearemos un objeto denominado *Network Manager* que contendrá un *script* que se encargará de conectarnos al servidor en cuanto empiece la aplicación. Para ello haremos uso de las funciones proporcionadas por PUN. En concreto, emplearemos las relacionadas con *Photon Realtime*. En este script crearemos, en el caso de que no exista, una sala con el nombre de "Sala de pruebas" y estableceremos las opciones de dicha sala. Entre las opciones que nos permite modificar PUN se encuentran el número de jugadores por sala, si esta es visible y si está abierta.

A continuación procederemos a mostrar a los otros jugadores de la sala. Para ello añadiremos otro *script* al *Network Manager*. Este nuevo *script* será el encargado de instanciar un *prefab* del jugador cuando acceda a la sala y de destruirlo cuando abandone la misma. Denominaremos a este *prefab* como *Network Player*.

Network Player deberá de tener como hijos una esfera que representará la cabeza y dos esferas más pequeñas que representarán las manos, simulando así el cuerpo de los jugadores. Además, deberá tener un componente *Photon View*, que nos permitirá identificar el objeto en la red, y un *script* que será el responsable actualizar la posición de la cabeza y las manos según la posición del visor y de los controladores. Los hijos de este, por su parte, deberán tener un

componente *Photon Transform View* para que los datos acerca de sus transformaciones se envíen por la red.

Por último, haremos que el jugador pueda interactuar con los cubos pequeños. Para conseguirlo será necerario que estos tengan como componente el *script OVRGrabbable* que nos proporciona Oculus Integration, un *Box Collider* y un *Rigidbody*. Con esto el jugador ya podrá coger y mover los cubos. El siguiente paso es que ese movimiento se sincronice en la red. Para ello agregaremos a los cubos un componente *Photon View* y otro *Photon Rigidbody View*. Un aspecto a tener en cuenta es que únicamente el jugador que sea propietario del cubo en la red, es decir, el primer jugador que coja el cubo. Esto supone que el segundo jugador no podrá coger el cubo. Para cambiar esto será necesario modificar el componente *Photon View* del cubo y establecer el parámetro de *Ownership Transfer* como *Takeover*. Deberemos también modificar el *script* de *OVRGrabbable* para que al coger el cubo se solicite la propiedad de su *Photon View*. Una vez realizados dichos cambios ya tenemos la escena de prueba lista.

Como hemos comentado anteriormente, después de probar el funcionamiento de esta prueba se descartó el uso de Photon PUN 2 debido a la latencia alta y a que los cubos no se movían apropiadamente si los golpeabas con otro cubo. Esto último se debe a que cuando el cubo no es propiedad del jugador que lo golpea con otro, este no se mueve porque el *script* que hemos creado solicita la propiedad al cogerlo, pero no al golpearlo con otro objeto. Para esto se han realizado pruebas de cambiar la propiedad cuando se golpea con otro objeto pero no funcionaban correctamente y el cubo se movía de manera incorrecta. Para nuestro videojuego este último aspecto es muy importante, ya que el jugador deberá de golpear el disco por medio del *pusher*.

6.2.2. Normcore

De nuevo partiremos de la escena base que habíamos comentado anteriormente. Para poder empezar a emplear Normcore en nuestro proyecto será necesario descargarlo e importarlo al proyecto desde la *Asset Store* de Unity.

Una vez importado, deberemos de meter en nuestra escena el *prefab Realtime + VR Player* que nos proporciona Normcore. En este encontramos un campo denominado *App Key*. Para que funcione el sistema de Normcore será necesario registrarnos en su página web y obtener

una *App Key* para introducir en ese campo.

El siguiente paso será rellenar los campos del *script* que contiene *Realtime + VR Player*. Para completarlos será necesario arrastrar el objeto *OVRPlayerController* dentro del campo *Root*, *CenterEyeAnchor* en el campo *Head*, *LeftHandAnchor* en *Left Hand* y *RightHandAnchor* en *Right Hand*. Esto nos servirá para que la cabeza y las manos del jugador se actualicen con el movimiento del visor y de los controladores. Como podemos observar en este *script*, Normcore ya nos proporciona un *prefab* para los jugadores, que contiene una malla para la cabeza y otra para las manos.

Una vez configurada la conexión, pasaremos a configurar los objetos para que el jugador pueda cogerlos y moverlos. En este caso, los cubos no estarán directamente en la escena, sino que instanciaremos uno cada vez que un jugador acceda a la sala. Para ello deberemos crear un *prefab* del cubo que contenga la malla, un *Rigidbody*, un *Box Collider*, el *script OVRGrabbable* que nos proporciona Oculus Integration[1] y los *scripts Realtime Transform* y *Realtime View* que nos proporciona Normcore. En este último deberemos de desmarcar la opción de *Owned by Creating Client*. Además, deberemos de crear un *script* al que denominaremos *GrabRequest* para detectar cuando un jugador está cogiendo el cubo y solicitar su propiedad, para que así pueda coger dicho cubo.

Por último, crearemos un *script* para que cuando nos conectemos a una sala se instancie un cubo de los que acabamos de crear. Será necesario que agreguemos este *script* como componente del *Realtime + VR Player*. Cabe destacar que, a diferencia de Photon PUN 2, al incluir en nuestra escena el *Realtime + VR Player* nos conectamos por defecto a una sala al iniciar la aplicación.

Tras finalizar esta escena de prueba y comprobar su funcionamiento, decidimos que Normcore sería mejor opción para nuestro videojuego, ya que al pasar el cubo entre dos jugadores se veía más fluido y con menos *lag* que la prueba anterior realizada con Photon PUN 2.

6.3. Implementación

Para realizar el desarrollo del juego partiremos de la prueba realizada con Normcore descartando la parte de los cubos, que serán sustituidos por un disco y dos *pushers*.

[1] Aquellos *scripts* o *prefabs* que empiecen por OVR son proporcionados por Oculus Integration.

6.3.1. Escena Lobby

Como Normcore no dispone de herramientas para la gestión de salas y la creación del *lobby*, haremos uso de una base de datos de MongoDB. Esta nos permitirá también limitar las salas a dos personas. Para poder comenzar a utilizarla será necesario registrarnos en MongoDB y crear una colección para almacenar los documentos correspondientes a cada una de las salas. Para cada sala únicamente almacenaremos cuatro valores: el identificador que proporciona automáticamente MongoDB, el nombre de la sala, el número de jugadores en la sala y el nombre del jugador que la creó. Este id lo emplearemos como nombre de sala al conectarnos con Normcore, para así evitar colisiones de nombres en las salas. Por otro lado, el nombre de la sala y el del jugador que la creó lo utilizaremos para mostrarlo en la tabla de salas disponibles del *lobby*. Por último, el número de jugadores se emplea para dos cosas: mostrar solo las salas con un jugador y para meter en la sala a los dos jugadores cuando este número sea igual a dos.

Crearemos un objeto denominado *DatabaseManager* que se encargará de gestionar la conexión y las peticiones a la base de datos. Este contendrá un *script* que nos permitirá conectarnos a la base de datos, y más concretamente a la colección que hemos creado, listar todas las salas disponibles, crear una nueva sala, comprobar su estado, actualizarla y borrarla. En este *script* se ha establecido además un *try-catch* para que en caso de que no se pueda conectar con la base de datos muestre una pantalla de aviso al jugador para que revise su conexión a Internet.

Al comenzar el videojuego nos encontramos en la escena de *lobby*, que es local para cada jugador. Esta escena contendrá las interfaces necesarias para la creación y selección de salas, además de almacenar los datos del jugador que serán necesarios para la escena de la partida. Estos datos los almacenaremos en un objeto denominado *PlayerInfo*. Este contendrá el nombre del jugador, el id de la sala que este ha creado o seleccionado y un entero que servirá para saber en qué posición deberá de aparecer en la escena del juego. A este último valor lo denominaremos *spawn* y lo emplearemos también para identificar al jugador que tenga un 0 como *player1* y el que tenga un 1 como *player2*. Esta clase será un *singleton* para poder acceder a ella fácilmente desde cualquier parte del código.

Primero comenzaremos a preparar la escena para la creación de las interfaces y la interac-

ción del jugador con las mismas. Para ello introduciremos un *canvas* en nuestra escena, que modificaremos poniéndolo en la capa por defecto, cambiando el modo de render a *World Space* e introduciendo como evento de cámara el objeto *CenterEyeAnchor*, que es hijo del objeto *OVRPlayerController*. Además, le agregaremos el *script OVRRaycaster* para que funcione correctamente con nuestro visor. Dentro de este *canvas* iremos creando los botones con los que el usuario podrá interactuar.

Para que el jugador pueda interactuar con estos deberemos incluir el *script OVRPhysicsRaycaster* en el objeto *OVRCameraRig*. Además, deberemos agregar a nuestra escena el *prefab UIHelpers*, que nos proporcionará un sistema de eventos que dispone de un componente denominado *OVRInputModule*. En este *script* arrastraremos el objeto *RightHandAnchor* al campo *Ray Transform* para indicar que el controlador derecho será el puntero láser y el que lanzará los eventos de *input*. Deberemos activar el componente *Line Renderer* del objeto *Laser Pointer*, que nos proporciona también *UIHelpers*, para que el jugador pueda ver el rayo que sale del controlador y con el que va a interactuar con los botones. Para presionar un botón de nuestra interfaz, por tanto, deberá de apuntar hacia él con el puntero láser y presionar el *botón A* del controlador derecho.

El *canvas* que hemos preparado antes contendrá cinco interfaces distintas: *PlayerNameUI*, *RoomsAvailableUI*, *CreateRoomUI*, *WaitingUI* y *ConnectionFailsUI*. Cada una de estas interfaces será un objeto de la escena, que contendrán a su vez los elementos visuales de la misma.

La primera interfaz con la que interactuará el jugador al comenzar el videojuego será con la de *PlayerNameUI*. En esta el jugador deberá de introducir su nombre por medio de un teclado de VR. Para crear dicho teclado deberemos de crear un botón para cada letra, uno para el espacio, un botón para borrar todo, uno para borrar una letra y otro de *enter*. Además, tendrá un texto arriba donde iremos mostrando lo que el jugador escriba con nuestro teclado. Cuando el jugador presione el botón de *enter*, siempre y cuando haya completado el campo, se almacenará dicha información en el objeto *PlayerInfo*.

Figura 6.6: Prototipo de la pantalla de PlayerNameUI dentro del juego

Una vez introducido su nombre, el jugador pasará a la interfaz *RoomsAvailableUI*. Como bien indica el nombre, esta mostrará una lista con *scroll* de las salas disponibles. Estas salas se obtienen por medio de la clase *DatabaseManager*, que realiza una petición a la base de datos para que le proporcione todas las salas cuyo campo número de jugadores sea igual a uno. Esta interfaz se encargará de instanciar un botón para cada una de las salas y almacenará en él el id de la sala, el nombre de la sala y el del creador. Estos dos últimos datos serán los que aparezcan escritos en el botón, mientras que el id se empleará para almacenarlo como id de la sala en el objeto *PlayerInfo* cuando el jugador lo pulse. Al pulsarlo se le almacenará también un uno como *spawn* en *PlayerInfo*, por lo que este será el jugador 2. En el caso de que al pulsar esa sala ya hubiera cambiado a dos jugadores en la base de datos, se volvería a solicitar la lista de salas para mostrar las nuevas salas disponibles y no se actualizaría *PlayerInfo*. En caso contrario, se cambiará al jugador de escena a la sala con la mesa de Air Hockey.

Para evitar la ralentización del juego, en lugar de estar actualizando continuamente la lista de salas disponibles, estableceremos un botón para poder actualizar dicha lista únicamente cuando el jugador pulse en él.

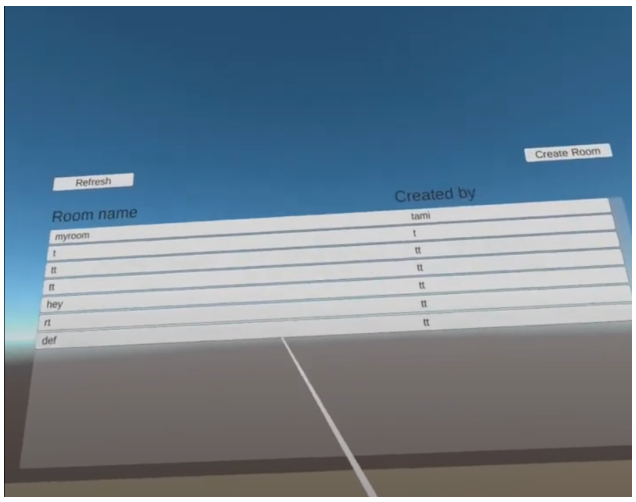


Figura 6.7: Prototipo de la pantalla de RoomsAvailableUI dentro del juego

En esta pantalla también encontraremos un botón que al pulsarlo nos llevará a la interfaz para crear salas, *CreateRoomUI*. En esta encontraremos un teclado idéntico al explicado anteriormente, en el que el jugador deberá de escribir el nombre de la sala que desea crear, y un botón para volver a la pantalla anterior. Cuando el jugador presione el botón de *enter*, siempre y cuando haya completado el campo, se hará uso de la clase *DatabaseManager* para almacenar la nueva sala en la base de datos. En la base de datos se almacenará, por tanto, el id autogenerado por MongoDB, el nombre de la sala que acabamos de escribir, un uno como número de jugadores en la sala y el nombre del creador que lo obtendremos de *PlayerInfo*.

Figura 6.8: Prototipo de la pantalla de CreateRoomUI dentro del juego

Después de pulsar *enter* se redirigirá al jugador a otra pantalla donde deberá de esperar a que otro jugador seleccione su sala. En esta pantalla aparecerá el texto *"Waiting for an opponent"* para que el jugador sea consciente de que está esperando a un oponente para empezar la partida. A esta interfaz la denominaremos *WaitingUI* y se encargará de comprobar de forma asíncrona en cada update si el número de jugadores en la sala que ha creado es igual a dos por medio de una petición a la base de datos. De este modo, cuando se cumpla dicha condición se eliminará la sala de la base de datos, se almacenará el id de la misma y un cero como *spawn* en *PlayerInfo*, de modo que este será el jugador 1, y se cambiará al jugador de escena a la sala con la mesa de Air Hockey. Por otro lado, esta pantalla también dispone de un botón para volver a la pantalla con la lista de salas. En el caso de que el jugador decida darle a ese botón mientras está esperando al otro jugador, se eliminará la sala que ha creado de la base de datos para que no le aparezca a otros jugadores. Este botón se bloqueará una vez pulsado para que el jugador no pueda lanzar varias veces la misma petición.

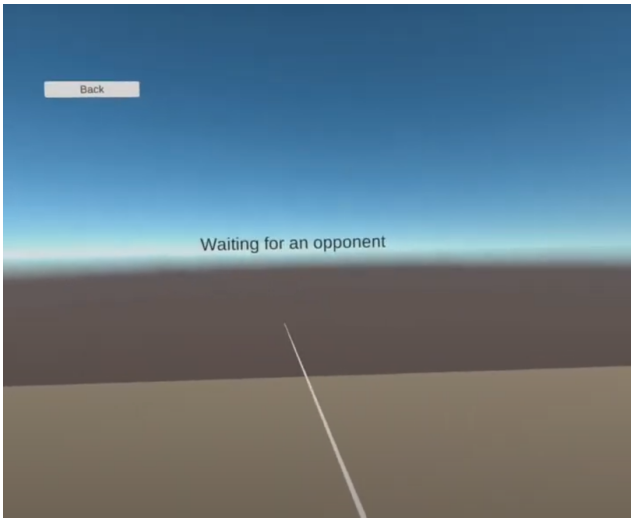


Figura 6.9: Prototipo de la pantalla de WaitingUI dentro del juego

Por último encontramos la interfaz de *ConnectionFailsUI* que, como hemos comentado anteriormente, aparecerá cuando una de las peticiones a la base de datos falle porque el juego no se ha podido conectar con la misma.

Figura 6.10: Prototipo de la pantalla de ConnectionFailsUI dentro del juego

Toda la gestión de pantallas se hace gracias al objeto de *UIManager*, que permite cambiar de una pantalla a otra llamando a la función que genera la pantalla a la que queremos acceder. Esta clase es *singleton*, por lo que podremos acceder a ella desde cualquier *script* sin necesidad de instanciar dicha clase.

6.3.2. Escena Air Hockey

En esta escena encontramos un objeto mesa que tendrá la malla de nuestra mesa de Air Hockey y varios *Box Collider* colocados de manera que el disco no pueda atravesar la misma. Estos colisionadores, al igual que el disco y el *pusher*, tendrán un material físico[2] para simular la escasa fricción que tenemos en una mesa de Air Hockey real. El disco y el *pusher* serán dos *prefabs* que instanciaremos cuando los jugadores se conecten a la sala de Normcore. Deberemos configurar estos de la misma manera que hicimos con los cubos en la prueba del multijugador de Normcore. Es decir, cada *prefab* contendrá la malla correspondiente, un *Rigidbody*, un *Mesh Collider*, el *script OVRGrabbable*, los *scripts Realtime Transform* y *Realtime View* y el *script GrabRequest* que habíamos creado. En este caso, a diferencia de los cubos anteriores, pondremos un *Mesh Collider* en lugar de un *Box Collider* para que se ajuste a la forma cilíndrica de estos. Además, el disco contendrá un *script* para que rebote

[2]Los materiales físicos en Unity determinan cómo un colisionador interactúa con los otros.

correctamente al colisionar con los márgenes de la mesa. Este *script* se encargará, por tanto, de calcular el ángulo en el que deberá de rebotar el disco y de recalcular la velocidad del mismo.

Para configurar esta escena necesitaremos los datos almacenados en *PlayerInfo*. Es por ello que deberemos de crear un *script* para no eliminar dicho objeto al cambiar de escena. Para ello, primero tendremos que colocarle un *tag* al objeto *PlayerInfo* y crearle como componente el *script DontDestroy*. Después, en este *script* haremos que en la función *Awake*[3] busque en la escena todos los objetos con el *tag* que hemos puesto antes y que ejecute la función *DontDestroyOnLoad* con el objeto *PlayerInfo* que habrá encontrado en la búsqueda anterior.

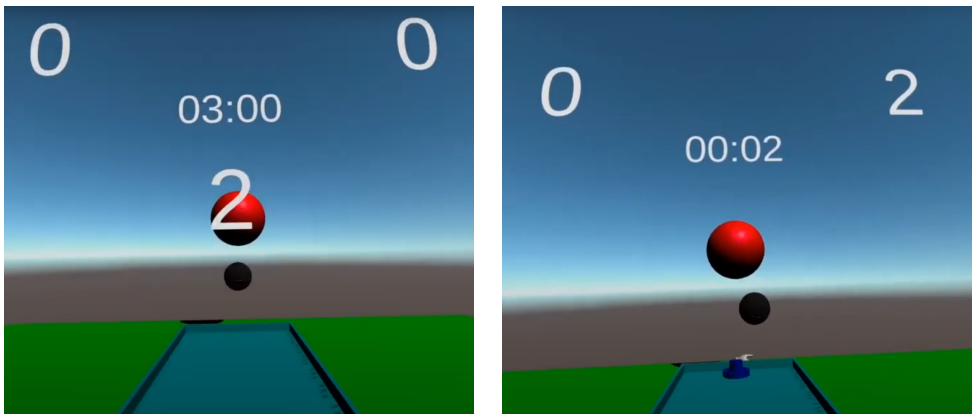
Emplearemos el id de la sala de *PlayerInfo* como nombre para crear la sala de Normcore, ya que al ser único evitaremos colisiones de nombres. Una vez conectados, dependiendo de si su *spawn* es 0 o 1, colocaremos al jugador a un lado u otro de la mesa de Air Hockey e instanciaremos el *pusher* del color que corresponda (rojo si es 0 y azul si es 1). Recordamos de nuevo que el jugador cuyo *spawn* sea 0 será el jugador 1 y el otro el jugador 2.

Además, esta escena contará con tres interfaces distintas a las que denominaremos *ScoreObjects*, *TimeObjects* y *EndScreens*. Al igual que en la escena anterior, estas interfaces estarán contenidas dentro de un *canvas*, que deberemos de configurar como hemos explicado antes.

La primera interfaz, *ScoreObjects*, contendrá dos textos para la puntuación de cada jugador. Estas puntuaciones están repetidas porque situaremos dos marcadores en la escena. Estos se situarán en la parte superior de la mesa frente a cada jugador, de manera que cada uno verá el marcador situado en su lado de la mesa.

Con la interfaz de *TimeObjects* sucede lo mismo que en el caso anterior, solo que en este caso tenemos un texto para mostrar la cuenta atrás y otro para mostrar el tiempo de la partida. El tiempo de partida se situará en la parte superior de la mesa, como en las mesas de Air Hockey originales, mientras que la cuenta atrás se situará a la altura de la vista del jugador y será más grande. De nuevo, cada jugador tendrá estos textos enfrente.

[3]La función Awake la tienen por defecto todos los objetos de Unity y se utiliza para inicializar variables o estados del juego antes de que este empiece.



(a) Cuenta atrás activa **(b)** Partida empezada

Figura 6.11: Prototipo de las interfaces ScoreObjects y TimeObjects dentro del juego

Por último, y siguiendo la estructura de los textos repetidos para cada jugador, encontramos la interfaz de *EndScreens*. Esta contendrá un texto para indicar al jugador que ha finalizado la partida y un botón para volver atrás. Este botón funcionará igual que los comentados para la escena de *lobby*. Cuando el jugador lo pulse, se hará reset del objeto *PlayerInfo*, manteniendo únicamente el nombre del jugador, y se volverá a cargar la escena de *lobby*.

Figura 6.12: Prototipo de la interfaz EndScreens dentro del juego

A continuación, crearemos el objeto *GameManager* que será el encargado de controlar en todo momento los distintos estados del juego por medio de dos *scripts*:

- ***ScoreSync:*** su función principal es sincronizar la puntuación de ambos jugadores. Para ello deberemos de crear también la clase *ScoreSyncModel*. Esta clase heredará de *RealtimeModel*, una clase que nos proporciona Normcore para poder sincronizar datos personalizados durante la partida. En concreto, *ScoreSyncModel* almacenará el nombre de cada uno de los jugadores y las puntuaciones de los mismos. Por otro lado, *ScoreSync* heredará de la clase *RealtimeComponent<ScoreSyncModel>*. *RealtimeComponent* creará por defecto una propiedad de la clase llamada modelo que se completará con una instancia de nuestro *ScoreSyncModel*. De esta manera, al realizar algún cambio en los campos de la propiedad modelo, gracias a los eventos que nos proporciona Normcore y que nos avisan de cuando un valor del modelo ha cambiado, este se verá reflejado en la partida de ambos jugadores.

  Nada más conectarse a la sala de Normcore, cada jugador deberá llamar a la función de *ScoreSync* para guardar su nombre en el campo correspondiente del modelo, dependiendo de si se trata del jugador 1 o del jugador 2. Además, este *script* se encargará en la función *Start*[4] de seleccionar los textos de la interfaz *ScoreObjects* que van a estar activos, dependiendo de nuevo de si se trata del jugador 1 o del jugador 2. Esto se debe a que al estar las puntuaciones en la parte superior de la mesa frente a cada jugador, estos solo tendrán visibles dos de estos textos y no será necesario actualizar aquellos que no pueden ver. Para evitar que ambos jugadores puedan modificar los valores del modelo y se produzcan errores, será solo el jugador 1 el que pueda actualizar el modelo cuando se meta un gol. De esta manera, cuando el modelo cambie se lanzará el evento correspondiente y se actualizará el texto del marcador para ambos jugadores, manteniéndose así sincronizado en todo momento. Por último comentar que este *script* contiene una función que nos devuelve el nombre del ganador, en función de la puntuación de cada jugador, o *DRAW!* en caso de empate.

- ***TimerSync:*** su función principal es sincronizar el tiempo de ambos jugadores. De nuevo deberemos de crear una clase *TimerSyncModel* que herede de *RealtimeModel*. Esta almacenará un float que utilizaremos tanto para una cuenta atrás antes de comenzar

[4]Start, al igual que Awake, es una función que nos la proporciona por defecto Unity. La función Start se llamará una sola vez antes del primer update.

la partida como para el tiempo de la misma, un booleano para controlar cuando se ha conectado el jugador 1 y otro para el jugador 2, un booleano para saber si la partida ya ha comenzado y otro para saber si ha acabado. En este caso, *TimerSync* heredará de *RealtimeComponent<TimerSyncModel>*.

Nada más conectarse a la sala de Normcore, cada jugador deberá de modificar el booleano del modelo que indica que ya se ha conectado a Normcore. Dependiendo de si se trata del jugador 1 o 2 modificará un booleano u otro. Al igual que en el caso anterior, este *script* se encargará en la función de *Start* de seleccionar los textos de las interfaces *TimeObjects* y *EndScreens* que van a estar activos, dependiendo de si se trata del jugador 1 o del jugador 2. Para evitar de nuevo que haya problemas porque ambos jugadores actualizan la misma información, será otra vez el jugador 1 el encargado de actualizar el modelo. El proceso de actualización será el siguiente:

1. En la función *Update*[5] se comprobará que ambos jugadores se hayan conectado a Normcore, por medio de los booleanos correspondientes, antes de realizar ningún otro cambio en el modelo.

2. Una vez estén ambos jugadores conectados y mientras el juego no haya terminado, el jugador 1 irá descontando del *float* para el tiempo el *deltaTime* que pasa entre un frame y otro. Este valor de *deltaTime* nos lo proporciona la clase *Time* de Unity. En un primer momento, este tiempo se empleará para hacer una pequeña cuenta atrás antes de iniciar la partida (3, 2, 1, GO!). Gracias a los eventos de Normcore que hemos comentado anteriormente, cada vez que el modelo cambie se lanzará el evento correspondiente y se actualizará el texto que se les muestra por pantalla a ambos jugadores.

3. Una vez finalizada esa cuenta atrás, se establecerá el tiempo de partida en 3 minutos, se instanciará el disco y se pondrá a verdadero el booleano de que ha empezado la partida. De nuevo, al cambiar del modelo el booleano del inicio de partida se lanzará el evento de Normcore asociado a dicho parámetro. Este evento ocultará el texto en pantalla de la cuenta atrás para ambos jugadores. Ahora cada vez que el

[5]La función Update también nos la proporciona Unity por defecto. Esta se llama una vez por frame.

valor para el tiempo cambie, se actualizará el texto del tiempo en formato *mm:ss*, gracias a la clase *TimeSpan* de Unity.

4. Cuando este tiempo llegue a 0, se considerará que la partida ha finalizado y se pondrá dicho booleano a verdadero. Al cambiar dicho valor del modelo, el evento correspondiente se encargará de comprobar que el tiempo haya acabado y mostrará la pantalla de fin para cada jugador. Como hemos comentado anteriormente, en esta aparecerá el nombre del ganador, que lo obtenemos al llamar a la función de *ScoreSync* anteriormente comentada.

Para controlar cuando un jugador marca gol, colocaremos dos *Box Collider*, cada uno situado en una de las porterías. Este objeto tendrá un *script* que sobreescribirá la función de *OnCollisionEnter* de Unity. Cuando esta función se lance se comprobará que la colisión haya sido con el disco y, en caso afirmativo, se actualizará la puntuación correspondiente y se recolocará el disco en el lado de la mesa del jugador al que le acaban de marcar gol.

Colocaremos también un *Box Collider* en el suelo. De esta manera, si la colisión se produce con el disco o con uno de los *pushers*, los volverá a reposicionar en la mesa. En el caso del *pusher*, recolocará cada uno en su lado correspondiente dependiendo de si pertenece al jugador 1 o al jugador 2, mientras que si se trata del disco lo reposicionará en el lado de la mesa más cercano a donde ha caído. Esto se realiza debido a que los jugadores no se podrán mover de su sitio y, por tanto, no podrán ir a recogerlo.

Por otra parte, para evitar que el jugador salga despedido al colisionar el disco o el *pusher* con su cuerpo, tendremos que poner el cuerpo del jugador, es decir *OVRPlayerController*, en una capa y el disco y los dos *pushers* en otra. De esta manera, si configuramos que dichas capas no sean colisionables, por medio del apartado de *Layer Collision Matrix* en el menú de Physics, evitaremos este problema.

El siguiente paso será modificar el modelo que nos proporciona Normcore para el jugador, es decir, el *prefab VR Player*. El primer cambio será agregar un texto para mostrar el nombre de ese jugador encima de su cabeza. Para que este se mueva junto con la cabeza del jugador, el texto deberá ser hijo del objeto cabeza. En un primer momento este texto estará vacío para no mostrar nada a los jugadores hasta que se complete con el nombre real. Este se completará una vez se hayan conectado los dos jugadores a la sala de Normcore, obteniendo los nombres

del *script ScoreSync*.

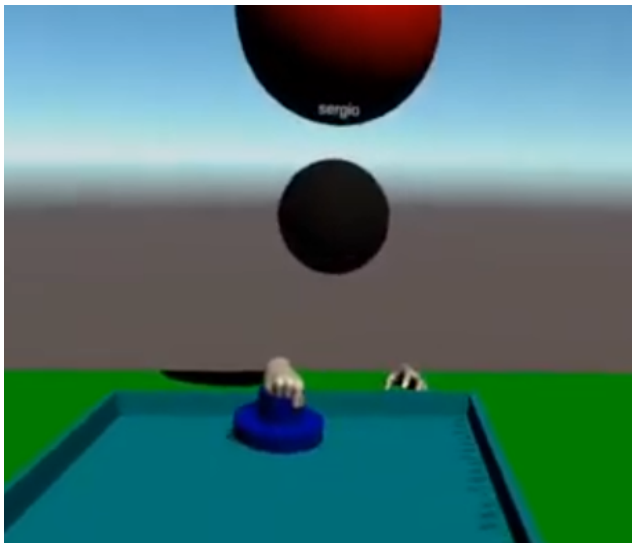


Figura 6.13: Nombre del jugador encima de su cabeza dentro del juego

A continuación cambiaremos las manos que vienen por defecto en *VR Player*. Para ello emplearemos el paquete de Unity de Valem [68], que contiene unas manos con animaciones de cerrar el puño y pellizcar. Este paquete viene preparado para usarlo con Photon PUN 2, por lo que deberemos de realizar algunos cambios para adaptarlo a Normcore. Para cambiar entre una animación y otra de las manos, deberemos de sincronizar los valores de *grip* y *trigger* que utiliza por medio de un *Realtime Model* de Normcore. Al igual que en los casos anteriores, crearemos la clase *HandAnimationModel*. Esta almacenará dos floats, uno para *grip* y otro para *trigger*. Crearemos otra clase denominada *HandAnimation* que heredará de *RealtimeComponent<HandAnimationModel>*. Esta clase se encargará de mantener sincronizadas las manos para ambos jugadores, actualizando los valores de *grip* y *trigger* en función de los botones de los controladores que hayan sido pulsados. Cuando el jugador presione el botón de *Trigger* del controlador, la mano asociada a este hará la animación de pellizcar, mientras que si pulsa el botón de *Grip* cerrará el puño. Estos botones los podemos ver identificados en los controladores de la figura 4.4 que hemos mostrado anteriormente.

Una vez sincronizada la animación de las manos, pasaremos a cambiar el agarre del disco y de los *pushers*. El primer paso será crear un objeto vacío como hijo de estos y ajustar su posición y rotación para colocarlo donde queremos que se sitúe la mano del jugador. Después

deberemos de colocar este objeto que acabamos de crear en el campo *Snap Offset* del *script OVRGrabbable* del objeto padre. Por último, tendremos que modificar el *script OVRGrabber* que nos proporciona Oculus Integration para permitir que el punto de agarre sea un hijo del objeto, como hemos explicado arriba, en lugar de un objeto separado de la escena.

Para finalizar, controlaremos que ningún jugador se quede solo en la partida antes de que esta finalice, de manera que si un jugador abandona la partida esta se dará por finalizada. Para controlar esto sobreescribiremos la función *AvatarCreatedDestroyed* que nos proporciona *Realtime* de Normcore. De este modo, cuando se destruya el avatar del jugador que ha abandonado la partida, se invocará esta función y esta se encargará de colocar el booleano de partida finalizada del *script TimerSync* a verdadero. Como hemos comentado anteriormente, al cambiar un valor del modelo se lanza el evento asociado a dicho cambio. En este caso, el evento asociado al cambio del booleano de partida finalizada, comprueba si el tiempo de partida restante es mayor que cero. En caso afirmativo, en lugar de mostrar el nombre del ganador como hemos indicado antes, mostrará un mensaje al jugador indicándole que el otro jugador ha abandonado la partida y el mismo botón de antes para volver al lobby.

Figura 6.14: Prototipo de la interfaz EndScreens cuando el otro jugador abandona la partida dentro del juego

6.4. Mejora gráfica e incorporación de audio

6.4.1. Iluminación

Tanto en esta escena como en la sala de air hockey será de noche, por lo que la iluminación será tenue. Sin embargo, en la escena de lobby la iluminación será mayor, tendrá un tono amarillo y procederá de una lámpara situada en una esquina de la habitación. Esto se debe a que el jugador en esta escena tendrá más tiempo para observar su entorno, por lo que al estar más iluminada el jugador podrá fijarse mejor en los detalles del escenario.

Por lo que respecta a la escena de la partida, la iluminación se situará principalmente sobre la mesa de air hockey, ya que es en esta donde se desarrollará la partida y donde queremos centrar el foco de atención. La luz del ambiente será menos intensa que en el caso anterior y tendrá un tono azulado, para conseguir ese estilo *gamer* que comentábamos anteriormente.

6.4.2. Modelado

Para realizar la mejora gráfica, el primer paso será modelar los elementos del *lobby* y de la sala con el air hockey.

Como hemos comentado anteriormente, el *lobby* imitará una sala de espera con dos puertas: una que simula la salida del juego y otra la entrada a la partida. Encima de cada una de estas puertas encontraremos un texto que imita las luces de neón, para así introducir parte de la estética *gamer* que podremos encontrar en la siguiente escena. Además, contaremos con una pantalla en la pared sobre la que mostraremos las diferentes interfaces.

Modelaremos otros elementos para completar el resto de la habitación, como puede ser una ventana, un sofá, lejas, una mesa de café o un aparador, entre otros.

Figura 6.15: Render de la escena de Lobby. Cámara enfocando a los sofás.

Figura 6.16: Render de la escena de Lobby. Cámara enfocando a la puerta de room.

Figura 6.17: Render de la escena de Lobby. Cámara enfocando a la puerta de exit.

Figura 6.18: Render de la escena de Lobby. Cámara enfocando al aparador.

En la figura 6.17 podemos observar un póster con el cartel del juego. Para la creación de

este se ha simulado una partida entre dos jugadores y se ha situado la cámara como si fuera los ojos de uno de ellos. Con ello se pretende que el observador se sienta inmerso en el cartel, como si fuera él el que está jugando.

Figura 6.19: Cartel del juego

En cuanto a la escena de la partida, en el centro de la habitación encontraremos nuestra mesa de air hockey, que será la principal protagonista. Para su modelado nos inspiraremos en las mesas típicas de las salas recreativas con el marcador arriba. Deberemos mejorar también el modelado del disco y de los *pusher* que estábamos empleando.

Para decorar el resto del escenario modelaremos máquinas arcade, un texto que simula luces de neón, un toca discos, una televisión, un sofá y otros elementos decorativos para

completar la escena.

Figura 6.20: Render de la escena de Air Hockey. Cámara enfocando a las máquinas arcades.

Figura 6.21: Render de la escena de Air Hockey. Cámara enfocando al sofá.

Figura 6.22: Render de la escena de Air Hockey. Cámara enfocando a los pósteres.

Figura 6.23: Render de la escena de Air Hockey. Cámara enfocando a las lejas.

Figura 6.24: Render de la escena de Air Hockey. Cámara enfocando al aparador.

Figura 6.25: Render de la escena de Air Hockey. Cámara desde la perspectiva del jugador 1.

Figura 6.26: Render de la escena de Air Hockey. Cámara desde la perspectiva del jugador 2.

Para algunas de las texturas creadas emplearemos imágenes procedentes de Freepik [69], las cuales se mencionan en el Anexo I.

Para el avatar de los jugadores emplearemos el casco de Hockey modelado por **Andrey Novichkov** como cabeza.

Figura 6.27: Casco de Hockey modelado por Andrey Novichkov

Por último, para resaltar que un jugador ha marcado gol, haremos que el disco explote al entrar en la portería. Para ello fracturaremos el modelo del disco en Blender, obteniendo así piezas más pequeñas del mismo. Después agruparemos todas estas piezas en un único objeto en Unity, que será el que instanciemos cuando marquen gol. Para simular la explosión deberemos de ponerle a cada pieza un *rigidbody* y un *mesh collider*. De esta manera, podremos aplicarle fuerzas a cada una de las partes del disco para que salgan despedidas por encima de la mesa. Transcurridos dos segundos, el disco fracturado se eliminará de la escena.

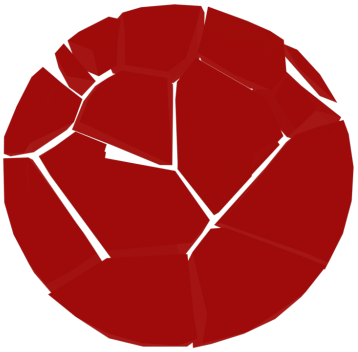


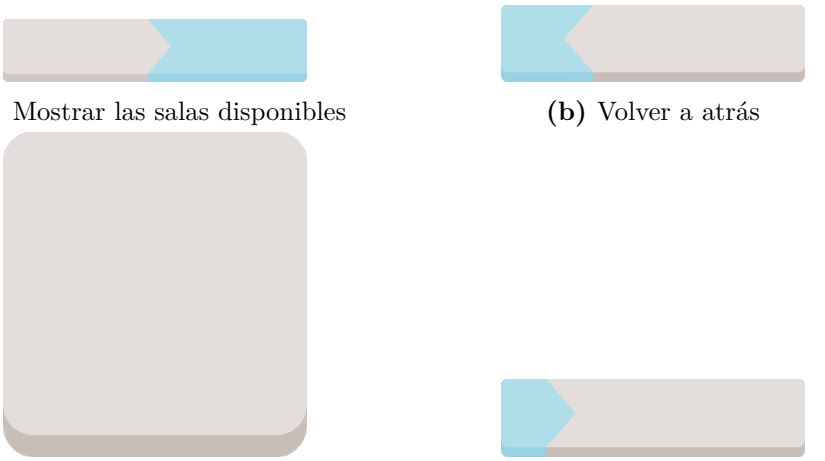
Figura 6.28: Disco fracturado

6.4.3. Interfaces

Para la creación de las interfaces seguiremos una estética común, donde los colores protagonistas serán el beige y el celeste que se muestran en la figura 6.29.

Figura 6.29: Color beige y celeste empleados en la interfaz

Crearemos cuatro tipos de botones distintos para distinguir visualmente su funcionalidad.



(a) Mostrar las salas disponibles

(b) Volver a atrás

(c) Teclado virtual

(d) Resto de funcionalidades

Figura 6.30: Tipos de botones diseñados y su funcionalidad

A continuación mostraremos el aspecto final de las pantallas anteriormente comentadas con los botones acabamos de mostrar:

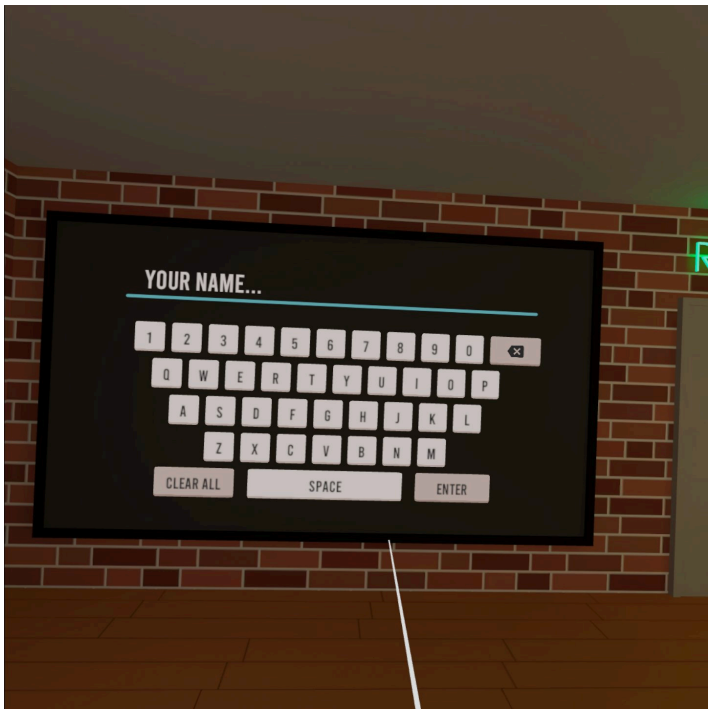


Figura 6.31: Interfaz de PlayerNameUI

Figura 6.32: Interfaz de RoomsAvailableUI

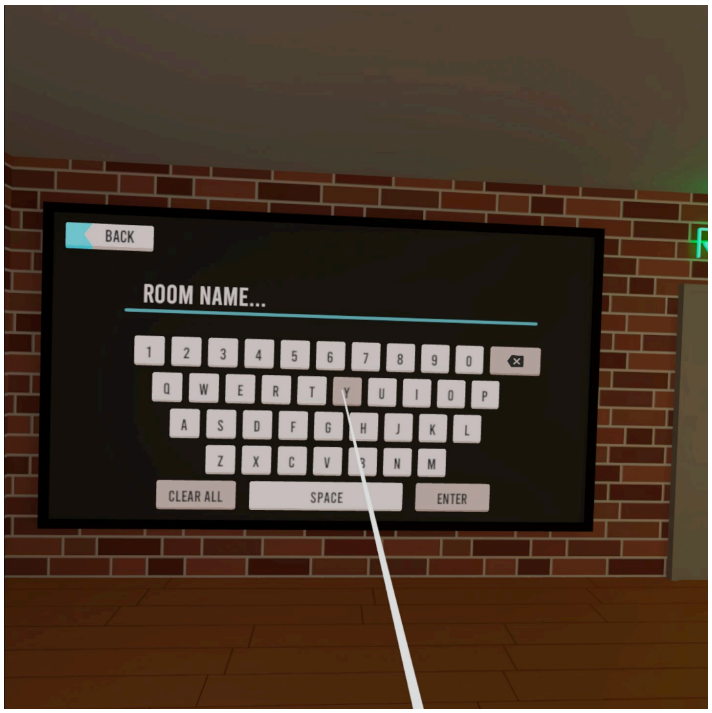


Figura 6.33: Interfaz de CreateRoomUI

Figura 6.34: Interfaz de WaitingUI

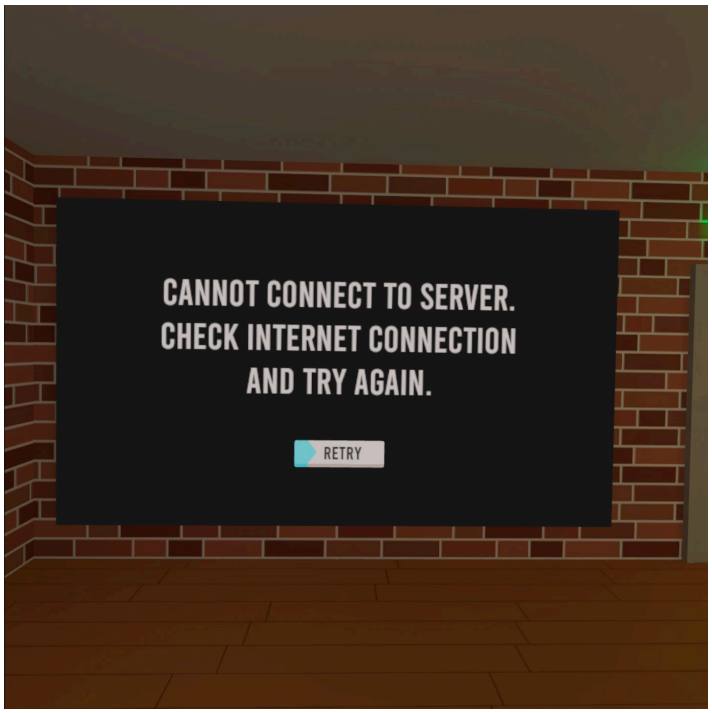


Figura 6.35: Interfaz de ConnectionFailsUI

En la pantalla de *WaitingUI*, que se muestra en la figura 6.34, podemos observar un icono de carga. Unity no admite formatos .gif, por lo que para darle movimiento a nuestro icono crearemos un *script* que vaya rotando la imagen cada cierto tiempo. De esta manera conseguiremos imitar el efecto cíclico de los archivos .gif.

Figura 6.36: Interfaz de TimeObjects con la cuenta atrás activada

Por lo que respecta al marcador, colocaremos la puntuación correspondiente a cada jugador del mismo color que su *pusher*, es decir, el jugador 1 tendrá el *pusher* rojo y sus goles aparecerán de color rojo. De esta manera conseguiremos que cada jugador identifique rápidamente cuál es su puntuación y la de su adversario.

En cuanto al nombre del adversario, este no aparecerá sobre la cabeza del mismo, sino que estará integrado en la parte delantera del casco. Para ello será necesario darle curvatura al texto para que se ajuste correctamente a la forma del casco.

Figura 6.37: Interfaz de ScoreObjects y TimeObjects con la partida empezada

Figura 6.38: Interfaz de EndScreens con la partida finalizada

Figura 6.39: Interfaz de EndScreens cuando un jugador abandona la sala

6.4.4. Música y efectos de sonido

En el Anexo I podrán encontrar información detallada acerca de las canciones y efectos de sonido empleados en este proyecto.

Por lo que respecta al *lobby*, aprovechando que hemos modelado un portátil con altavoces, haremos que la música provenga del mismo. Para ello crearemos un objeto en dicha posición y le pondremos el componente *Audio Source*. En este componente podremos colocar el fichero de audio de nuestra canción y ajustar el volumen. Además, deberemos colocar el campo *Spatial Blend* en 3D y ajustar la distancia máxima a la que se seguirá escuchando la música en el apartado de *3D Sound Settings*.

Como hemos comentado anteriormente, para esta escena emplearemos música del género Lo-fi. En concreto emplearemos la canción **Sad Thoughts** de Glitch Cloud. El ritmo de esta canción ayudará a que el tiempo de espera antes de entrar en partida sea más ameno para el jugador.

Por contra, al entrar en partida queremos que el ritmo de la canción sea más energético para que encaje con el ritmo de juego. Es por ello que emplearemos la canción **Smile** de LiQWYD.

En este caso, la música no será 3D, sino que tendrá el mismo volumen independientemente del lugar de la sala donde te encuentres. Esto se debe a que no queremos que el jugador note cambios en el volumen de la música mientras juega.

En cuanto a los efectos de sonido, partiremos del audio de una partida de air hockey real. De este extraeremos los fragmentos necesarios y reduciremos el ruido. Necesitaremos un audio que simule el golpe del disco con el *pusher*, otro para el golpe entre el disco y la mesa y el sonido que hace el disco al caer después de marcar gol. Estos tres audios serán un componente *Audio Source* de nuestro disco y en el *script* de este haremos que en la función *OnCollisionEnter* reproduzca uno de los audios anteriores, dependiendo del objeto con el que el disco haya colisionado. Para estos sonidos estableceremos de nuevo el campo *Spatial Blend* en 3D.

Por último, pondremos otro efecto de sonido para el momento en el que el jugador marque gol. Este sonido consistirá en una bocina que se reproducirá cuando se actualice la puntuación del marcador en el *script* ScoreSync.

7. Conclusiones

Para concluir este TFG cabe destacar que se han cumplido todos los objetivos planteados al comienzo del mismo. Es por ello que AirHockey VR puede considerarse un videojuego completo y acabado.

Después de llevar a cabo una investigación sobre los motores gráficos que permitían RV, decidimos desarrollar el proyecto con Unity, ya que consideramos que era el más apropiado para iniciarse en el mundo de la RV debido a la gran comunidad de desarrolladores y el amplio número de tutoriales. Por otra parte, para seleccionar el sistema de multijugador se realizaron pruebas de eficiencia entre los sistemas disponibles. El sistema seleccionado fue Normcore, debido a la menor latencia y mejor interacción entre objetos. Como consecuencia de dicha selección creamos nuestro propio sistema de gestión de salas mediante una base de datos de MongoDB, ya que este sistema no disponía de uno propio. Si bien es cierto que la selección de Normcore nos permitió alcanzar un mejor rendimiento en el videojuego, la escasez de documentación y foros sobre el mismo dificultó en algunas ocasiones el proceso de desarrollo.

Durante el proceso de implementación de las mecánicas, el objetivo principal era conseguir simular las físicas de una mesa de Air Hockey real. Después de varias pruebas, este objetivo se logró alcanzar mediante un *script*, ajeno al cálculo de las físicas de Unity, que calcula la dirección a la que debería rebotar el disco al colisionar. Este añadido mejora la experiencia de usuario y permite lograr un resultado más realista.

Una vez finalizado el primer prototipo de nuestro videojuego, fue imprescindible realizar una fase de pruebas para encontrar posibles fallos en el desarrollo. Esto nos permitió asegurar el correcto funcionamiento del videojuego antes de pasar a la fase de mejora gráfica.

El modelado de los escenarios y su incorporación en el juego cambiaron por completo la estética del juego, mejorando la experiencia de usuario. Gracias a la iluminación se consiguió

diferenciar las escenas de *lobby* y la sala de air hockey, pero manteniendo una estética similar en ambas para que los jugadores asocien que se tratan de habitaciones de un mismo edificio.

Por último destacar la importancia de incorporar música y efectos de sonido al juego. La música diferenciada en cada una de las escenas permite que el jugador manifieste emociones diferentes. Mientras que la canción del *lobby* transmite tranquilidad, la canción de la partida transmite todo lo contrario. Esta última es una canción más enérgica y que hará que el jugador esté más motivado durante la partida. Por otro lado, los efectos de sonido nos ayudan a informar al jugador sobre qué está ocurriendo en la partida.

7.1. Trabajo futuro

Si bien es cierto que AirHockey VR es un videojuego completo y acabado, durante las fases de desarrollo han surgido nuevas ideas que se podrían implementar en un futuro. Algunas de estas ideas son las siguientes:

- Crear y entrenar una IA que permita jugar en modo *single player*.
- Simular otros juegos característicos de las salas recreativas, como es el caso del billar o del futbolín. De esta manera, en la lista de salas disponibles no solo se mostraría el nombre de la sala y quien la ha creado, sino también el modo de juego.
- Crear unas estadísticas personales donde se le muestre al jugador el número de partidas jugadas y el número de victorias conseguidas. En el caso de haber distintos modos de juego, estos valores se mostrarían para cada uno de ellos.
- Incluir la posibilidad de crear salas privadas y acceder mediante código. De esta manera, un jugador podrá crear una sala privada y se le proporcionará un código. Este código es el que deberá de compartir con sus amigos para que accedan a la sala que acaba de crear.
- Crear un menú de ajustes que permita al usuario cambiar algunas opciones como el idioma, el volumen de la música y el de los efectos de sonido.
- Subir el videojuego a la plataforma SideQuest VR para obtener *feedback* por parte de usuarios externos a mi círculo.

Referencias

[1] Teseo. Aplicaciones y usos de la realidad virtual. https://teseo.es/noticias/aplicaciones-y-usos-de-la-realidad-virtual/.

[2] Mae Anderson. Realildad virtual no fue alivio para encierros de covid-19. https://www.latimes.com/espanol/eeuu/articulo/2021-06-01/realildad-virtual-no-fue-alivio-para-encierros-de-covid-19.

[3] SL. Los beneficios de la realidad virtual contra el encierro durante la pandemia. https://www.clarin.com/tecnologia/beneficios-realidad-virtual-encierro-pandemia_0_vKaI-VF3u.html.

[4] Tokio School. Tendencias realidad virtual: desarrollos futuristas. https://www.tokioschool.com/noticias/tendencias-realidad-virtual/.

[5] Mireia Luisa Sempere Tortosa. Introducción a la realidad virtual. Material de la asignatura de Realidad Virtual curso 2020-2021 del Grado en Ingeniería Multimedia de la Universidad de Alicante.

[6] Solidvirtual (InnXR). 6 diferentes usos de la realidad virtual en sectores útiles. https://www.helptoengineering.com/es/blog/6-diferentes-usos-de-la-realidad-virtual-en-sectores-utiles.

[7] Computer World. La realidad virtual, una industria por explotar. https://www.computerworld.es/tecnologia/la-realidad-virtual-una-industria-por-explotar.

[8] Eva Rodríguez De Luis. Guía de compra de gafas de realidad virtual: 11 modelos para todas las expectativas, necesidades y presupuestos. https:

//www.xataka.com/seleccion/guia-compra-gafas-realidad-virtual-16-modelos-para-todas-expectativas-necesidades-presupuestos.

[9] Zeiss vr one plus en amazon. https://www.amazon.es/ZEISS-VR-ONE-realidad-smartphone/dp/B01GHN88HU.

[10] Google cardboard. https://arvr.google.com/cardboard/get-cardboard/.

[11] Oculus rift s gafas de realidad virtual en pccomponentes. https://www.pccomponentes.com/oculus-rift-s-gafas-de-realidad-virtual.

[12] Htc vive cosmos v2 gafas de realidad virtual en pccomponentes. https://www.pccomponentes.com/htc-vive-cosmos-v2-gafas-de-realidad-virtual.

[13] Htc vive pro full kit realidad virtual en pccomponentes. https://www.pccomponentes.com/htc-vive-pro-full-kit-realidad-virtual.

[14] Htc vive cosmos elite gafas de realidad virtual en pccomponentes. https://www.pccomponentes.com/htc-vive-cosmos-elite-gafas-de-realidad-virtual.

[15] Valve index en steam. https://store.steampowered.com/valveindex.

[16] Hololens 2 en microsoft. https://www.microsoft.com/es-es/p/holoLens-2/91pnzzznzwcp/?activetab=pivot%3aoverviewtab.

[17] Hp reverb g2 gafas de realidad virtual en pccomponentes. https://www.pccomponentes.com/hp-reverb-g2-gafas-de-realidad-virtual.

[18] Nintendo labo kit de vr - set de expansiÓn 1 en game. https://www.game.es/VIDEOJUEGOS/UTILIDAD/NINTENDO-SWITCH/NINTENDO-LABO-KIT-DE-VR-SET-DE-EXPANSION-1/164174#descripcion.

[19] Playstation vr v2 + cámara + vr worlds en fnac. https://www.fnac.es/Playstation-VR-V2-Camara-VR-Worlds-descargable-Accesorios-informatica-Gafas-VR/a6309677.

[20] Correa elite para quest 2. https://www.oculus.com/accessories/quest-2-elite-strap/.

[21] Sidequest. https://sidequestvr.com/.

[22] Oculus quest 2. https://www.oculus.com/quest-2/.

[23] Estuche link para quest 2. https://www.oculus.com/accessories/quest-2-carrying-case/.

[24] Cable oculus link para quest 2. https://www.oculus.com/accessories/oculus-link/.

[25] David Rodríguez. Oculus quest 2 establece un nuevo récord con más de 1 millón de dispositivos vendidos en su primer trimestre. https://www.hobbyconsolas.com/noticias/oculus-quest-2-establece-nuevo-record-1-millon-dispositivos-vendidos-primer-trimestre-803087.

[26] Impresiones 3d para oculus quest 2 en thingiverse. https://www.thingiverse.com/search?q=oculus+quest+2&type=things&sort=relevant.

[27] Impresiones 3d para oculus quest 2 en etsy. https://www.etsy.com/es/search?q=oculus%20quest%202.

[28] Vrcompare - compare vr headset specs, prices and reviews. https://vr-compare.com/.

[29] Ángel Morán Santiago. Los mejores juegos de realidad virtual que puedes jugar ahora mismo. https://www.hobbyconsolas.com/listas/mejores-juegos-realidad-virtual-actuales-785459.

[30] Half-life: Alyx en steam. https://store.steampowered.com/app/546560/HalfLife_Alyx/.

[31] Beat saber en steam. https://store.steampowered.com/app/620980/Beat_Saber/.

[32] Astro bot rescue mission. https://store.playstation.com/es-es/product/EP9000-CUSA12392_00-PLATFORMERVR00EU.

[33] Tetris effect. https://www.tetriseffect.game/buy/.

[34] Dan Griliopoulos Vic Hood, Mark Knapp. Best vr games 2021: the top virtual reality games to play right now. https://www.techradar.com/best/the-best-vr-games.

[35] Superhot vr en steam. https://store.steampowered.com/app/617830/SUPERHOT_VR/.

[36] Arizona sunshine en steam. https://store.steampowered.com/app/342180/Arizona_Sunshine/.

[37] Trover saves the universe en steam. https://store.steampowered.com/app/1051200/Trover_Saves_the_Universe/.

[38] Eleven table tennis. https://www.oculus.com/experiences/quest/1995434190525828/?intern_source=blog&intern_content=best-of-quest-2020-a-look-back-at-the-years-top-vr-games-and-apps.

[39] Sports scramble. https://www.oculus.com/experiences/quest/705576999566582/.

[40] The climb. https://www.oculus.com/experiences/quest/2376737905701576/?intern_source=blog&intern_content=best-of-quest-2020-a-look-back-at-the-years-top-vr-games-and-apps.

[41] Hyper dash. https://www.oculus.com/experiences/quest/3070239359662935/.

[42] Cubism. https://www.oculus.com/experiences/quest/2264524423619421/.

[43] Puzzling places - beta. https://www.oculus.com/experiences/quest/3623167214470921/?utm_source=sidequest.

[44] Proyectos Agiles. Qué es scrum. https://proyectosagiles.org/que-es-scrum/.

[45] Wikipedia. Air hockey. https://en.wikipedia.org/wiki/Air_hockey.

[46] Wikipedia. Oculus quest. https://en.wikipedia.org/wiki/Oculus_Quest.

[47] Wikipedia. Oculus quest 2. https://en.wikipedia.org/wiki/Oculus_Quest_2.

[48] Iván Asensio. Mejores motores grÁficos para desarrollar videojuegos. https://www.soloempleo.com/motor-grafico-para-programar-videojuegos.

[49] Wikipedia. Unity (motor de videojuego). https://es.wikipedia.org/wiki/Unity_(motor_de_videojuego).

[50] Unity - nuestra empresa. https://unity.com/es/our-company.

[51] Wikipedia. Unreal engine. https://es.wikipedia.org/wiki/Unreal_Engine.

[52] Dejan Gajsek. Unity vs unreal engine for xr development: Which one is better? https://circuitstream.com/blog/unity-vs-unreal/.

[53] Bolt visual scripting. https://assetstore.unity.com/packages/tools/visual-scripting/bolt-163802.

[54] Adeyemi Thompson. Unity3d vs unreal engine – which is the right engine for you? https://sbanimation.com/unity3d-vs-unreal-engine-which-is-the-right-engine-for-you/.

[55] Maciej Zareba. Unity vs unreal comparison: Choose the best engine for you. https://4experience.co/unity-vs-unreal-comparison-choose-the-best-engine-for-you/.

[56] Oculus integration sdk. https://developer.oculus.com/downloads/package/unity-integration/25.0/.

[57] Understand oculus integration package components. https://developer.oculus.com/documentation/unity/unity-utilities-overview/?locale=es_ES&device=QUEST.

[58] Multijugador y networking (redes) - unet. https://docs.unity3d.com/es/2018.4/Manual/UNet.html.

[59] Photon pun. https://www.photonengine.com/pun.

[60] Normcore. https://normcore.io/.

[61] Mongodb atlas. https://www.mongodb.com/cloud/atlas.

[62] Wikipedia. Mongodb. https://es.wikipedia.org/wiki/MongoDB.

[63] Blender. https://www.blender.org/.

[64] Wikipedia. Gimp. https://es.wikipedia.org/wiki/GIMP.

[65] Wikipedia. Inkscape. https://es.wikipedia.org/wiki/Inkscape.

[66] Audacity. https://audacity.es/.

[67] Get started with oculus in unity. https://developer.oculus.com/documentation/unity/unity-gs-overview/.

[68] Valem. https://www.patreon.com/ValemVR.

[69] Freepik. https://www.freepik.es/.

[70] David Méndez. Cagr. https://numdea.com/cagr.html#:~:text=La%20tasa%20de%20crecimiento%20anual,tres%2C%20o%20incluso%20diez.

[71] La factoria 3D. Servicio de fotogrametría. https://www.lafactoria3d.es/servicios/fotogrametria.

[72] Wikipedia. Videojuego casual. https://es.wikipedia.org/wiki/Videojuego_casual.

[73] Wikipedia. Videojuego independiente. https://es.wikipedia.org/wiki/Videojuego_independiente.

[74] Shah Anas. A step by step guide on making a game design document. https://www.tekrevol.com/blogs/making-a-game-design-document/.

[75] Rubén Soler Ferrer. Guía para crear un game design document. http://www.rubensolerferrer.com/guia-para-crear-un-game-design-document/.

[76] Wikipedia. Lo-fi. https://es.wikipedia.org/wiki/Lo-fi_(m%C3%BAsica).

Lista de Acrónimos y Abreviaturas

IA Inteligencia Artificial.

RV Realidad Virtual.

TFG Trabajo Final de Grado.

A. Anexo I - Créditos

En este apartado haremos atribución a los legítimos autores de todo el material no original contenido en este proyecto.

A.1. Imágenes

Algunas de las imágenes han sido diseñadas por Freepik:

- Cuadros grandes de la sala de Air Hockey (https://www.freepik.es/vector-gratis/coleccion-minimalista-portadas-japonesas_6951628.htm#page=1&query=art%20japanese&position=1)
- Marco de fotos de la leja de la sala de Air Hockey (https://www.freepik.es/vector-gratis/gente-dibujos-animados-coleccion-mascotas_14371182.htm#page=15&query=cartoon&position=13)
- Marco de fotos de la leja de la sala de Aihockey 1 (https://www.freepik.es/vector-gratis/paisaje-invierno-diseno-plano_10669193.htm#query=landscape%20cartoon&position=41)
- Póster de la sala de Air Hockey (https://www.freepik.es/vector-gratis/fondo-delicioso-sopa-ramen_9979164.htm#page=1&query=japanese&position=17)
- Póster de la sala de Air Hockey 1 (https://www.freepik.es/vector-gratis/coleccion-minimalista-portadas-japonesas_6968938.htm#page=1&query=japanese%20art&position=47)

- Póster de la sala de Air Hockey 2 (https://www.freepik.es/vector-gratis/retro-artilugios-carteles-dibujos-animados-camara-radio-reproductor-musical-telefono-tv-despertador_4016631.htm)
- Portada de un vinilo de la sala de Air Hockey (https://www.freepik.es/vector-gratis/cerezo-japones-dibujado-mano_839392.htm#page=1&query=japanese&position=21)
- Portada de un vinilo de la sala de Air Hockey 1 (https://www.freepik.es/vector-gratis/album-musica-electro_3601853.htm#query=MUSIC+cd&position=0)
- Portada de un vinilo de la sala de Air Hockey 2 (https://www.freepik.es/vector-gratis/impresion-artes-marciales_1389669.htm#page=5&query=japanese&position=22)
- Portada de un vinilo de la sala de Air Hockey 3 (https://www.freepik.es/vector-gratis/diseno-vinilo-cartel-festival-musica_9057680.htm#page=2&query=vinilo&position=46)
- Patrón lateral para una máquina arcade de la sala de Air Hockey (https://www.freepik.es/vector-gratis/patron-lineas-abstractas-planas-lineales_13685667.htm#page=1&query=pattern&position=36)
- Patrón lateral para una máquina arcade de la sala de Air Hockey 1 (https://www.freepik.es/vector-gratis/patron-semitono-triangular-diferentes-colores_6024585.htm#page=1&query=pattern&position=48)
- Cuadros grandes de la sala de Lobby (https://www.freepik.es/vector-gratis/coleccion-portadas-arte-abstracto-dibujado-mano_14350528.htm#page=1&query=art%20abstract&position=48)
- Marcos de fotos de la sala de Lobby (https://www.freepik.es/vector-gratis/coleccion-cubiertas-botanicas-oro_12300568.htm?query=art)
- Pantalla portátil de la sala de Lobby (https://www.freepik.es/vector-gratis/diseno-streaming-musica_4239657.htm#page=1&query=music%20player&position=

4)

- Caja de la leja de la sala de Lobby (https://www.freepik.es/vector-gratis/fondo-floral-dibujado-mano_3733427.htm#page=1&query=pattern&position=17)
- Caja de la leja de la sala de Lobby 1 (https://www.freepik.es/vector-gratis/elegantes-patrones-lineas-dibujadas-mano-cuatro-colores_8152311.htm#page=1&query=pattern&position=31)
- Caja del aparador de la sala de Lobby (https://www.freepik.es/vector-gratis/fisuras-patron-geometrico-inspiracion-japonesa_3841564.htm#page=1&query=pattern&position=2)

A.2. Modelados

Únicamente dos de los modelos empleados en este proyecto no son propios.

Por un lado encontramos las manos con las animaciones para el multijugador que se emplean en la sala de Air Hockey. Estas han sido realizadas por Valem y podemos encontrar en enlace de descarga en su canal de Youtube (https://www.youtube.com/watch?v=KHWuTBmT1oI&ab_channel=Valem).

Por otro lado, el casco que emplean los avatares ha sido creado por Andrey Novichkov. En la página de descarga se puede observar un comentario que realicé en el que le solicitaba permiso para su utilización y Andrey me lo concedió (https://sketchfab.com/3d-models/hockey-helmet-washington-capitals-e49ac53913ab4bb5af3396cd273ef783).

A.3. Sonidos

El videojuego usa estos sonidos de Freesound:

- "Air Hockey Distance" de portwain (https://freesound.org/people/portwain/sounds/260894/) con licencia CC0
- "130606_krupowki hockey" de Zbylut (https://freesound.org/people/Zbylut/sounds/191796/) con licencia CC BY-NC

Como efecto de sonido para cuando un jugador marca gol se ha empleado el sonido **basketball buzzer sound effect** de la página FreeSoundEffect (`http://freesoundeffect.net/sound/basketball-buzzer-sound-effect`).

Para la canción del lobby se ha empleado la canción **Sad Thoughts** de Glitch Cloud. En el siguiente enlace podemos ver como la autora permite su uso siempre que se le créditos: `https://www.youtube.com/watch?v=gF2Z7ZGUKfA&list=PL0ONFXpPDe_mtm3ciwL-v7EE-7yLHDlP8&index=14&ab_channel=GlitchCloud`.

Por último para la canción durante la patida se ha empleado **Smile** de LiQWYD:

> Smile by LiQWYD | https://soundcloud.com/liqwyd/ Music promoted by https://www.chosic.com/ Licensed under Creative Commons: Attribution 3.0 Unported (CC BY 3.0) https://creativecommons.org/licenses/by/3.0/

B. Anexo II - Descarga APK del videojuego

En el siguiente enlace podrán descargar el APK de AirHockey VR:

`https://drive.google.com/file/d/1XnXmWQIhtxmg5cRviFry7T_raX6Pfz4D/view?usp=sharing`

Para su instalación podrán hacer uso de la plataforma SideQuest VR o emplear Android SDK Platform Tools, entre otros.

B.1. Instalación vía SideQuest VR

El primer paso será descargar la aplicación de escritorio de SideQuest VR (**`https://sidequestvr.com/setup-howto`**). Una vez instalada, en el menú superior de la aplicación encontramos la opción de *Instalar APK desde una carpeta del ordenador*.

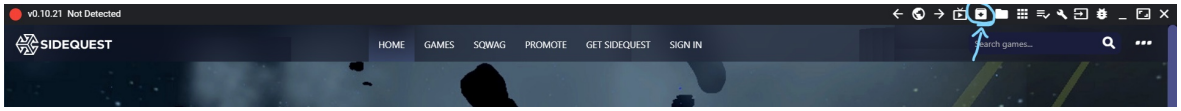


Figura B.1: Menú superior de la aplicación SideQuest VR con la opción de Instalar APK marcada

Esta opción abrirá un explorador de archivos donde deberá de seleccionar la APK de AirHockey VR que se acaba de descargar. Será necesario que las Oculus Quest estén conectadas al ordenador.

B.2. Instalación vía Android SDK Platform Tools

En primer lugar deberá descargar las Android SDK Platform Tools (**`https://developer.android.com/studio/releases/platform-tools`**). Una vez extraído el contenido del ZIP, el siguiente paso será introducir la APK de AirHockey VR en dicha carpeta.

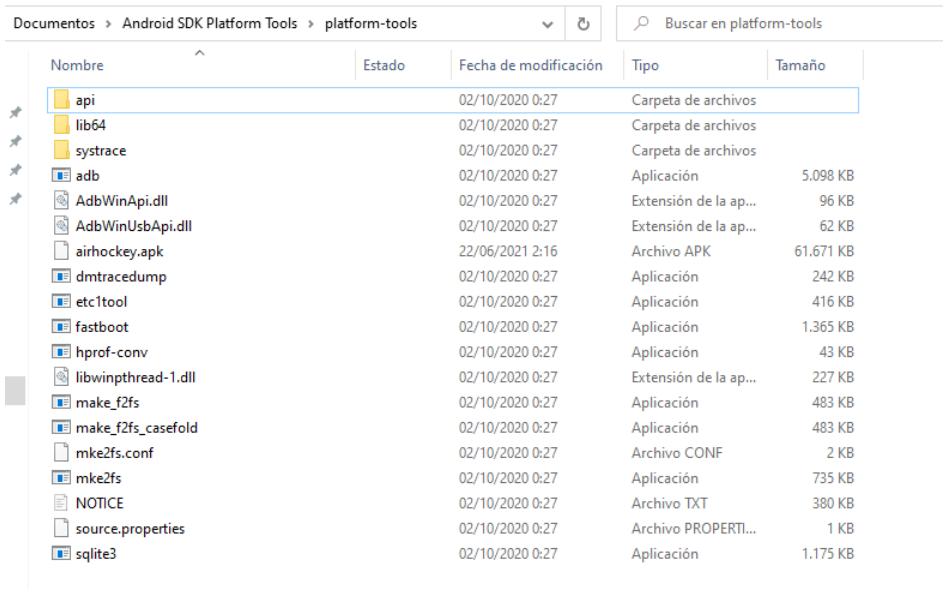


Figura B.2: APK de AirHockey VR dentro de la carpeta de Platform Tools

A continuación deberemos abrir una terminal en dicha carpeta y ejecutar el siguiente comando:

Código B.1: Comando para instalar APK

```
1 adb install airhockey.apk
```

De nuevo, será necesario que las Oculus Quest estén conectadas al ordenador.

B.3. Ejecución de la aplicación

Una vez instalada por uno de los métodos anteriores, deberá de lanzar la aplicación desde el menú de Oculus, en el apartado de aplicaciones de orígenes desconocidos.

Figura B.3: Menú de Aplicaciones de Oculus Quest con la opción de Orígenes desconocidos señalada

La aplicación de AirHockey VR le aparecerá la primera de la lista. Una vez seleccionada, podrá disfrutar del videojuego.